Trabajo de Fin de Grado

GRADO EN INGENIERÍA INFORMÁTICA

Facultad de Matemáticas e Informática
Universidad de Barcelona

IMPLEMENTACIÓN DE UN CLASIFICADOR DE PARTIDOS POLÍTICOS APLICADO A TWITTER

David Muntal Giménez

Director: Santi Seguí
Realizado en: Departamento de Matemáticas e Informática

Barcelona,1 de Febrero de 2019

Indice

1- Introducción al problema y motivación **2**

2 - Objetivos **4**

3 - Planificación **6**

4 - Arquitectura **8**

4.1 - Recopilación del conjunto de datos 9

4.2 - Tratamiento de los datos 12

4.3 - Clasificación de los datos 17

5 - Pruebas y resultados **29**

5.1 - Metodología 29

5.2 - Implementación 30

5.3 - Distribución de datos 33

5.4 - Clasificación 41

Conclusiones **59**

Referencias **60**

1- Introducción al problema y motivación

El problema trabajado durante este proyecto se centra en el campo del aprendizaje automático, o también conocido como ***Machine Learning***, y en concreto en el subdominio de aprendizaje supervisado. En este proyecto estudiaremos algunas de las técnicas más populares y las aplicaremos a un problema de clasificación de textos. En concreto, trabajaremos en un problema de detección de la ideología política de los textos de *tweets* a partir de su contenido.

La principal motivación para la realización de este proyecto es la comprensión y aplicación de las técnicas de **aprendizaje automático**. Un campo de trabajo muy popular en los últimos años donde mis conocimientos son, de un inicio, escasos y básicos.

Para estudiar estas técnicas hemos definido un problema propio: la clasificación de ideología política mediante texto. Para hacer frente a este problema, el proyecto consta de 4 partes principales:

- Recopilación del conjunto de datos.
- Tratamiento de texto mediante técnicas de lenguaje natural.
- Aplicación de técnicas de aprendizaje supervisado.
- Evaluación de los métodos propuesto mediante distintas pruebas.

Nos hemos decidido por un aprendizaje basado en *tweets* ya que, para llevar a cabo la clasificación, necesitamos una buena cantidad de datos de texto corto, que puedan ser obtenidos de un modo relativamente sencillo y que ofrezcan gran versatilidad en el uso de las palabras y significados. Además estos datos son públicos y de acceso prácticamente ilimitado.

En la plataforma *Twitter*, las principales formaciones políticas tienen presencia pública de mano de sus respectivos representantes, y esta se lleva a cabo mediante escritos cortos, haciendo uso de un límite de caracteres de texto (establecido por la propia plataforma *Twitter*).

Cada uno de estos textos (*tweets),* serán utilizados como datos de entrada para el entrenamiento del modelo clasificador de este proyecto.

Es posible que alguna de la información compartida como *tweet* vaya acompañada de imágenes, archivos de vídeo o enlaces a fuentes externas. En estos casos, un clasificador del tipo en que se centra este proyecto no resulta útil, de modo que será necesario revisar la información para desechar estos.

Partiendo de un número significativo de datos recogidos de inicio, y a pesar de que algunos vayan a ser ignorados dada su poca utilidad, confiamos en que el conjunto de datos final seguirá siendo una representación suficientemente fiel de cada grupo como para generar un conjunto de entrenamiento útil.

Considerando estos requisitos hemos decidido centrar el problema en clasificar dentro de 6 **partidos políticos** o grupos.

Con la dificultad que puede suponer la correcta interpretación de palabras clave, será importante conocer los detalles particulares de cada clasificador que utilicemos, y también los diferentes parámetros de los modelos, y cómo ajustarlos para generar la mejor predicción posible.

Observaremos este comportamiento en las pruebas.

2 - Objetivos

El objetivo fundamental de este proyecto es **la creación de un clasificador** que **pueda relacionar un *tweet* con el partido político al que es más probable que pertenezca**.

Partiendo de un conjunto de escritos (con un máximo de 280 caracteres), nuestro objetivo es establecer un sistema que gestione las palabras utilizadas, aplicar algoritmos de Aprendizaje Automático supervisado, y finalmente asignar una predicción (nombre del partido) a un nuevo *tweet* .

Como inicialmente hemos decidido tener en cuenta 6 partidos políticos, si realizamos una asignación aleatoria se consigue un $\sim 1/6 = 16.66\%$ de precisón.

Recopilación y tratamiento de los datos

El objetivo en este caso es, obtener los datos de los distintos partidos y encontrar el mejor subconjunto de palabras para construir el modelo de clasificación.
Será importante que las palabras elegidas proporcionen:

- Mayor discrepancia entre grupos.
- Mayor similitud dentro del grupo

También habrá que tener en cuenta los tipos de palabras, su tamaño, y en qué modo pueden flexionarse.

Por otra parte, también habrá que crear un sistema que permita transformar esos conjuntos de datos iniciales en la versión vectorizada de los datos más relevantes.

Para calcular el éxito obtenido en el tratamiento de los datos hace falta ver qué resultados se obtienen en la predicción, por lo que el éxito en este objetivo viene condicionado por el siguiente objetivo.

Determinación del tipo de clasificador

Hay muchos y muy distintos tipos de clasificadores, sin embargo en este proyecto nos centraremos en solamente cuatro.

El objetivo en este punto es **encontrar el clasificador** que mejor se adapte al tipo de datos que estamos tratando y ajustarlo para **obtener el mejor ratio de exactitud**.

Hay que tener en cuenta que el modo en que se traten los datos inicialmente implicará la creación de una cantidad mayor o menor de variables independientes.

Según el clasificador que se utilice, esto tendrá una repercusión importante en el porcentaje de acierto obtenido.

Teniendo en cuenta la complejidad del problema inicial, y las 6 posibles respuestas preestablecidas, consideramos que **un objetivo realista es el de conseguir un clasificador que otorgue un mínimo del 51% de exactitud**.

Dicho de otro modo, que de cada dos respuestas obtenidas, al menos una sea correcta.

3 - Planificación

Determinación de la viabilidad del proyecto (1 Semana)
Verificamos que es posible llevar a cabo el proyecto de clasificación, y definmos los detalles y lenguajes de programación.

Recopilación de información sobre clasificadores (3 Semanas)
Buscamos información sobre tipos de clasificadores, cómo funcionan y cómo se comportan los modelos con diferentes tipos de datos.

Recopilación de datos para clasificación (1 Semana)
Extracción y *limpieza* de los datos obtenidos de *Twitter,* (~1800 *tweets)*

Programación de la transformación de los datos (4 Semanas)
Determinamos cómo será el vector de datos finales (*hot encoding)* y programamos en *Python,* los diferentes métodos y variables para, reconocer y extraer los datos dentro de los rangos determinados. (Variables *MIN,MAX,MINP,MAXP, cw, cp)*
Programación en *R* del código para la extracción y *limpieza* de Train / Test

Pruebas y análisis de los datos transformados (2 Semanas)
Llevamos a cabo pruebas para asegurar y corregir la correcta extracción de los datos con los parámetros establecidos.

Programación de los clasificadores (4 Semanas)
Determinamos los clasificadores seleccionados y programación en *R* de estos.
Incluye transformaciones necesarias para ajustarse a los detalles de cada clasificador.
Eliminación de datos con: Desviación estándar cero. Alta correlación. Datos vacíos.
Transformaciones: Matriz Sparse, Normalización.

Ajustes en los clasificadores (3 Semanas)
Modificación de valores y recepción de datos para mejorar los resultados en exactitud y tiempo.
Pruebas con clasificadores (3 Semanas)
Diferentes pruebas para testear el éxito de clasificación y cómo mejorarlo

Corrección de Errores (1 Semana)
Eliminación de partes no necesarias, y ajustes para solucionar o mejorar problemas surgidos en la fase de pruebas.

Diagrama de Gantt

Actividad	Duración	Semanas																					
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Determinación de la viabilidad del proyecto	1 Semana																						
Recopilación de información sobre clasificadores	3 Semanas																						
Recopilación de datos para clasificación	1 Semana																						
Programación de la transformación de los datos	4 Semanas																						
Pruebas y análisis de los datos transformados	2 Semanas																						
Programación de los clasificadores	4 Semanas																						
Ajustes en los clasificadores	3 Semanas																						
Pruebas con clasificadores	3 Semanas																						
Corrección de Errores	1 Semana																						

4 - Arquitectura

A fin de poder aplicar algoritmos de clasificación a los datos es necesario reunir una cantidad de datos significativa (**4.1**) con la que poder entrenar y probar el clasificador.

A continuación, asegurar que esos datos son útiles y no contienen información innecesaria, es decir: hallar **la información más relevante** dentro de su grupo (**4.2**).

Este problema no es trivial ya que hay muchas de palabras y sus posibles combinaciones son inmensas, de modo que será de gran importancia ajustar bien los parámetros.

Seguidamente, codificar la información de estos *tweets* para convertir cada uno en vectores de datos numéricos manteniendo una representación de la información más significativa de ese *tweet* .

Finalmente, hará falta la creación de los subconjuntos de datos de Entrenamiento / Prueba (*Train / Test*) para aplicar el modelo clasificador (**4.3**) y observar los distintos resultados .

Por defecto hemos definido el una distribución de los datos 80% 20% respectivamente.

4.1 - Recopilación del conjunto de datos

En este punto, el objetivo es la obtención de los datos (*tweets)* de varios representantes políticos. La plataforma Twitter ofrece una API que permite el acceso al conjunto de tweets de cualquier cuenta pública. Sin embargo, no obtuvimos respuesta a la petición de acceso a esta API.

Afortunadamente, la plataforma ofrece un **buscador avanzado**, que hemos utilizado para extraer *tweets* de distintas cuentas, entre periodos de tiempo concretos.

A continuación se muestra el listado de cuentas de las que hemos descargado la información (en rango de fechas desde 1/03/2016 hasta 1/10/2018):

- Partido Popular
    - Xavier García Albiol - https://twitter.com/Albiol_XG
    - Mariano Rajoy - https://twitter.com/marianorajoy
    - Pablo Casado - https://twitter.com/pablocasado_
- Ciudadanos
    - Albert Rivera - https://twitter.com/Albert_Rivera
    - Inés Arrimadas - https://twitter.com/InesArrimadas
    - Carlos Carrizosa - https://twitter.com/carrizosacarlos
- Partido Socialista Obrero Español
    - Miquel Iceta - https://twitter.com/miqueliceta
    - Pedro Sánchez - https://twitter.com/sanchezcastejon
- Esquerra Republicana de Catalunya
    - Oriol Junqueras - https://twitter.com/junqueras
    - Gabriel Rufian - https://twitter.com/gabrielrufian
    - Marta Rovira - https://twitter.com/martarovira
- Junts per Catalunya
    - Carles Puigdemont - https://twitter.com/KRLS
    - Quim Torra - https://twitter.com/QuimTorraiPla
    - Elsa Artadi - https://twitter.com/elsa_artadi
- Podemos
    - Pablo Iglesias - https://twitter.com/Pablo_Iglesias_
    - Pablo Echenique - https://twitter.com/pnique
    - Irene Montero - https://twitter.com/Irene_Montero_

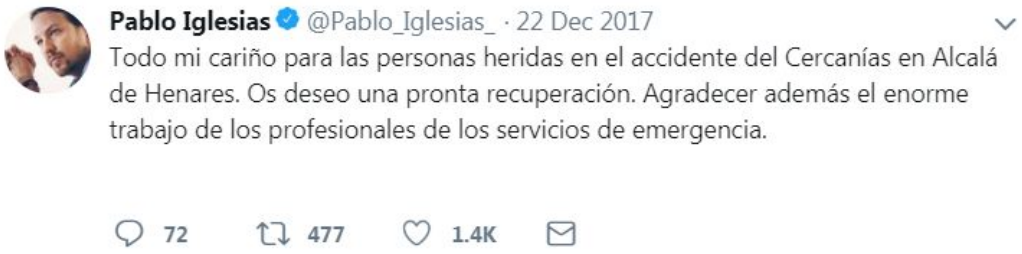


Imagen 1: Ejemplo de un *tweet* tal como se muestra en la plataforma

Para la extracción de datos, hemos tenido que hacerlo de un modo un poco rudimentario.

Primero buscamos los *tweets* de uno de los políticos entre las fechas determinadas.
Una vez cargados todos los datos en pantalla (**Imagen 1**), copiamos todo el contenido en un archivo de texto, y lo pasamos por un filtro programado en *Python.*

Creamos el filtro haciendo uso de *Expresiones Regulares (RegEx),* que reconocen el inicio y fin de cada *tweet*.

> Una *Expresión Regular* es una secuencia de caracteres que forma un patrón de búsqueda capaz de reconocer el tipo, cantidad y orden de los caracteres de texto que se están buscando.

Conociendo que, tal como se muestran por pantalla, cada tweet viene precedido por el nombre de la cuenta que lo publica (y detalles), y que al finalizar el tweet aparece: el número de respuestas, veces que has ido compartido, etc. Solamente es necesario crear dos *Expresiones Regulares* que reconozcan estos caracteres y guardar los datos (el *tweet* propiamente dicho) entre ambos casos.

Ejemplo del mismo *tweet* (**Imágen 1**) con los datos en *bruto* al ser copiados:

@Pablo_Iglesias_

22 Dec 2017

More

Todo mi cariño para las personas heridas en el accidente del Cercanías en Alcalá de Henares. Os deseo una pronta recuperación. Agradecer además el enorme trabajo de los profesionales de los servicios de emergencia.

72 replies 477 retweets 1,393 likes

Reply 72 Retweet 477 Like 1.4K Direct message

Aplicando el siguientes *RegEx* se reconoce la primera línea resaltada en rojo.

"(More)+([a-zA-Z0-9]){0,}"

Aplicando el siguientes *RegEx* se reconoce la línea final resaltada en rojo.

"d{0,3},?\d{0,3}? repl(y)?(ies)? \d{0,3},?\d{0,3}? retweet(s)? \d{0,3},?\d{0,3}? like(s)?"

Reconociendo ambas líneas, se extrae la información útil (resaltada en azul).

Cada uno de estos *tweets* se guardan como una sola línea de texto en un nuevo archivo con únicamente los datos necesarios.

También se utilizan *RegEx* para eliminar partes innecesarias del texto como por ejemplo enlaces a otras páginas.

Ejemplo de *RegEx* que reconoce direcciones URL:

"\b(https?|ftp|file)://[-A-Z0-9+&@#/%?=~_|!:,.;][A-Z0-9+&@#/%=~_|]"*

Para este proyecto hemos estimado oportuno, recoger un total aproximado de 18000 *tweets.* Manteniendo una proporción de ~3000 *tweets* por partido.

4.2 - Tratamiento de los datos

Dado que una palabra por sí sola y fuera de contexto, puede ser usada de modos completamente distintos, hemos considerado que probablemente ofrece más información una **pareja de palabras**, que **una sola palabra** sin contexto que la acompañe.

Sabiendo que muchas palabras con significados parecidos tienen una raíz común, y teniendo en cuenta que interesa más el significado que la palabra en sí, se utiliza un ***stem lematizador*** el cual simplemente reduce cada palabra del idioma español a su *lema*.

> El *lema* es la forma que por convenio se acepta como representante de todas las formas flexionadas de una misma palabra.

Por ejemplo, la palabra *decir* es el lema de *dije*, pero también de *diré* o *dijéramos.*
Las que usaremos de ahora en adelante son estas palabras lematizadas en minúscula.

Para empezar con la solución del problema, el programa lee todos los *tweets* pertenecientes a un mismo partido político. Y guarda una lista con cada **palabra / pareja de palabras usadas**, junto con el contador de veces que ese partido ha **usado** esa palabra / pareja. Finalmente conociendo el total de palabras usadas se extrae **el porcentaje de veces** que se ha usado cada palabra / pareja.

Este conteo se hace por cada partido. Para extraer el porcentaje de uso de la palabra dividimos la cantidad de veces que la ha usado un partido y la dividimos por el número de *tweets* de ese partido.

A continuación se definen los límites superior e inferior, a partir de los cuales se considera que las palabras o parejas, son suficientemente **buenos**.

Por ejemplo, si los valores que se establecen son: $MIN = 5$ y $MAX = 10$, se considerarán buenos datos las palabras, que hayan sido usadas un porcentaje de veces entre 5% y 10%

De este modo se pretende **evitar** tener en cuenta **palabras demasiado genéricas** que usarán todos los partidos (como artículos, determinantes, etc).

Y también **evitar palabras demasiado concretas** que puede que solo aparezcan en un *tweet* y que no representan un conjunto de datos relevante para clasificar.

Las variables MIN, MAX hacen referencia al rango en que se considera **buena** una **palabra.** Por otra parte, para las **parejas de palabras** se utilizan otros rangos ($MINP, MAXP$) dado que una pareja de palabras es mucho menos probable que aparezca con la misma asiduidad que una palabra sola, y prevemos que será necesario establecer otros rangos.

El ajuste de estos rangos definirá en gran medida la cantidad total de variables con las que tendrá que trabajar el modelo clasificador, por lo que nos resulta de gran importancia conocer y ajustar bien estos límites.

Una vez obtenida la lista de palabras y/o parejas de todos los partidos que se hallan dentro de los límites establecidos, se **aplica una nueva selección**, esta vez **teniendo en cuenta el número de partidos que, como máximo, han utilizado esa palabra / pareja**.

Para ello hacemos uso de las variables cw, cp destinadas a crear mayor separación entre clases.

- *cw* : El valor de esta variable le indica al programa el número de partidos que, **como máximo,** pueden haber utilizado esa **palabra**.
- *cp:* Funciona igual que *cw* pero para **parejas de palabras**

Veamos un ejemplo:

- La palabra *"Nunca"* la ha utilizado solo **1** partido. Nos referimos a ella como **A**.
- La palabra *"Justicia"* la han utilizado solo **3** partidos. Será **B**.
- La palabra "*Ahora"* la han utilizado **6** partidos (todos). Será **C**.

(Para este ejemplo suponemos que **A,B** y **C** tienen un porcentaje de uso dentro de los límites.)
Si utilizamos *cw = 1* en la selección de datos, la palabra **A** se reconocerá como palabra *buena*. Las palabras **B** y **C**, no.
Si utilizamos *cw = 3*, las palabras **A** y **B** se reconocen como palabras *buenas*. Pero **C** no.
Con *cw = 4*, las palabras **A** y **B** se siguen reconociendo como palabras *buenas*
ya que han sido usadas por, como máximo, *4* partidos políticos distintos
(**A** por 1 , **B** por 3) $\leq$ *4.* Pero **C** no.
Unicamente con *cw* = 6, **C** pasará a formar parte de las palabras *buenas.*

De este modo pretendemos deshacernos de los datos que aparezcan en todos los grupos, como *ruido*.

Estas palabras buenas serán las variables independientes en nuestro modelo.

Hay que considerar cómo se reducen la cantidad de variables independientes finales dependiendo de estos dos valores, y comprender cómo ayudan a la separación entre clases.

Un buen ajuste de estos parámetros debería ayudar a separar mejor los conjuntos y, con ello, ajustar los datos más relevantes para la clasificación.

Para encontrar cual es la mejor relación llevaremos a cabo pruebas con los distintos modelos.

Esta segunda selección dentro de los datos es una criba para resaltar aún más la singularidad de cada partido respecto del resto.

Observaremos cómo se comportan en las pruebas.

Codificación de los datos

Una vez consideremos que ya tenemos las palabras / parejas más relevantes para la clasificación, procedemos a la creación del vector de variables.

Primero se **indexan las palabras / parejas** "buenas" de **0 a *'n'*** (donde n es el número total de palabras / parejas). De modo que cada palabra "buena" tiene asociada un valor numérico único (entre 0 y 'n').

A continuación se crea un vector para cada tweet de tamaño *'n'* inicializando todas las posiciones a 0.
Luego se comprueba si ese *tweet* ha usado alguna palabra / pareja "buena" y en caso de ser así, se coloca en ese índice del vector el número de veces que ha usado esa palabra / pareja buena.

Ejemplo:
Si el *tweet* es *La verdad os hará libres y eso es verdad*
Y si las *palabras / parejas* "buenos" son los siguientes, y han sido indexados así:

0	1	2	3	4	5
'verdad'	'mentira'	'libre'	'coche'	'la verdad'	'ellos son'

El nuevo vector correspondiente a este *tweet* quedaría codificado de este modo:

0	1	2	3	4	5
2	0	1	0	1	0

- El índice 0 corresponde a la palabra "verdad" y su valor es 2 dado que en el texto sale 2 veces la palabra "verdad".

- Los índices 1, 3 y 5 tienen valor 0 dado que ninguna de esas palabras (“mentira”,”coche”,”ellos son”) estaba en el texto.
- El índice 2 tiene valor 1 dado que la palabra “libres” al ser reducida a su *lema* pasa a ser “libre” la cual sí que se encuentra en el conjunto de palabras “buenas”, y aparece 1 vez.
- El índice 5 tiene valor 1 ya que ha encontrado una pareja de palabras (‘la verdad’).

Finalmente se guardan todos los vectores de un mismo partido en un archivo en disco con el nombre del partido al que pertenecen.

Creación de Train/Test

Para crear el conjunto de datos totales, se carga de disco cada archivo (correspondiente a un partido político).

Seguidamente, se eliminan los vectores que tengan todos los valores a 0 (Puede pasar que algunos *tweets* sean muy cortos y no tengan ninguna de las palabras “buenas” encontradas en el paso previo (**4.2**), con lo que su vector resultante será todo 0 y no aporta información).

A continuación se añade el nombre de ese partido como clase (factor) en la primera columna de los vectores cargados.

Finalmente se extrae un porcentaje (generalmente 80% / 20%) de los datos de cada partido para crear el conjunto de datos *Train* y *Test*, respectivamente.
Estos se utilizarán para probar la exactitud del clasificador. Los datos *Train Test* se guardan en formato de dataset en archivos en disco.

Dado que muchos datos serán 0, hemos decidido no llevar a cabo la normalización de los datos para evitar que, debido al error en el redondeo de decimales, algunas variables aparezcan como constantes dentro de grupos, lo que supondría un problema para algunos clasificadores.

4.3 - Clasificación de los datos

La clasificación en *Machine Learning* [1],[2] es un tipo de aprendizaje supervisado. A partir de unos datos de entrenamiento de entrada se genera un modelo predictor que luego servirá para clasificar nuevas observaciones.

Dependiendo de cómo sean estos datos de entrenamiento (cantidad de observaciones, cantidad de variables) y el tipo de clasificación (binaria o multi-clase), resultará más efectivo un modelo u otro.

El caso concreto de este proyecto tendrá:

- Observaciones de entrenamiento: aproximadamente el 80% de un total de 18000.
- Número de variables: centenares o miles, dependiendo de la configuración en el tratamiento y preparación de los datos
- Tipo de clasificación: Multi-clase con 6 clases

Para la ejecución de este proyecto utilizaremos los siguientes modelos de clasificación supervisada.

- KNN
- LDA
- SVM
    - Lineal
    - No Lineal

k - Nearest Neighbors (KNN)

El algoritmo de clasificación *k - Vecinos Cercanos* funciona bajo el supuesto de que datos parecidos pertenecerán a la misma categoría.

Para calcular este parecido, en la fase de entrenamiento se genera un campo con tantas dimensiones como variables independientes tienen los datos.
A continuación se colocan las etiquetas de clase en el espacio n-dimensional utilizando los valores de entrenamiento como coordenadas de los puntos.

Para mostrar una representación visual simplificada (**Imagen 2**), se utilizan solamente dos variables ($X1, X2$), y un conjunto de etiquetas binario (blanco, negro).

En la fase de predicción, se colocan las nuevas observaciones en ese campo n-dimensional (**Imagen 3**), y se buscan los k puntos más próximos.

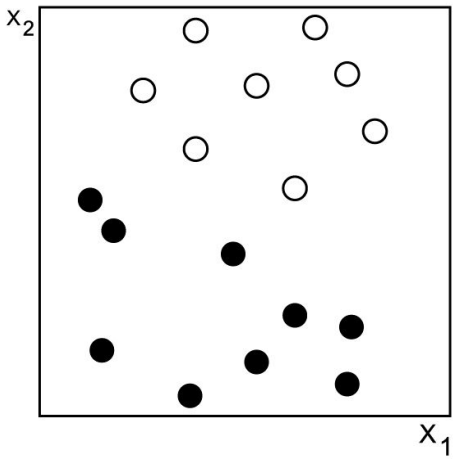


Imagen 2: Modelo creado con los datos de Entrenamiento

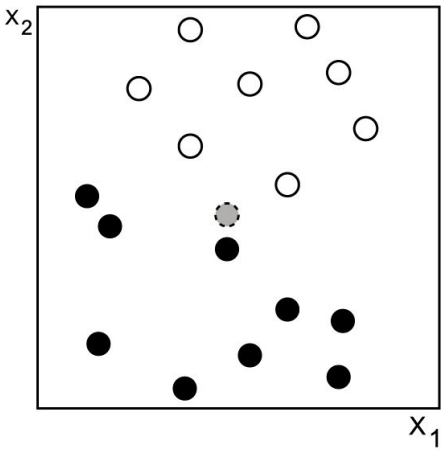


Imagen 3: Nuevo punto colocado en el espacio de entrenamiento

Estos k *vecinos,* al ser los más próximos, se considera que serán los más parecidos con lo que es más probable que pertenezcan a la misma clase que le punto que se pretende clasificar. La etiqueta más usada de entre estos k *vecinos* es la que se asigna al nuevo punto como predicción.

Ejemplos con diferentes valores para k.

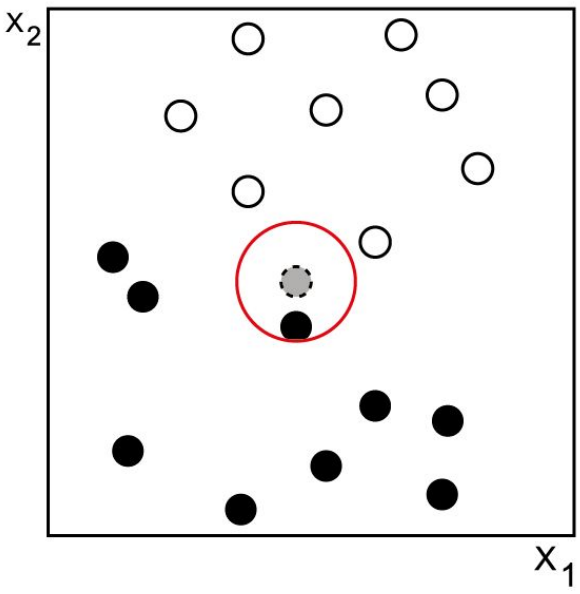


Imagen 4: Predicción con $k = 1$

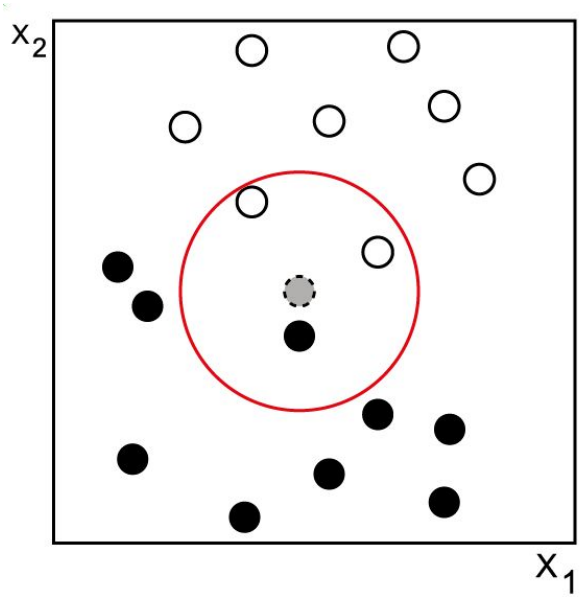


Imagen 5: Predicción con $k = 3$

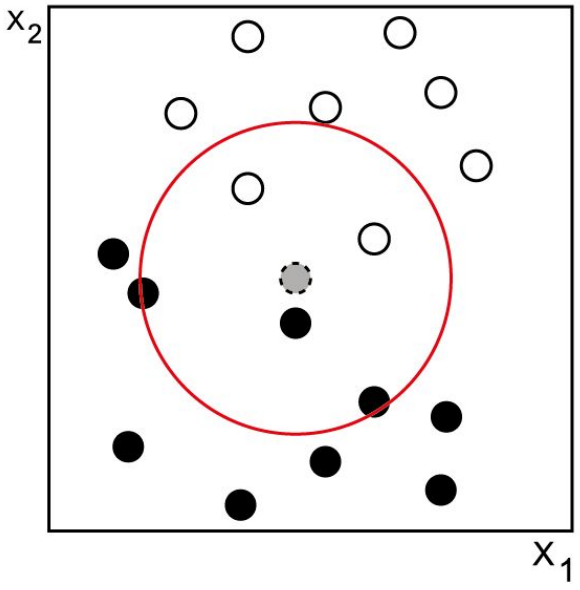


Imagen 6: Predicción con $k = 5$

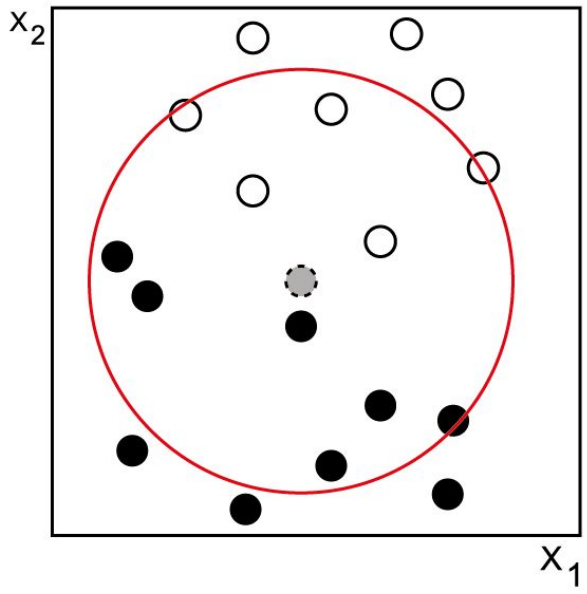


Imagen 7: Predicción con $k = 11$

Se puede observar que con distintos valores de *k* se obtienen predicciones distintas
En el ejemplo se clasifica con etiqueta "*Blanco*" si se usa $k = 3$ **(Imagen 5)**
Y "*Negro*" con $k = 1$ **(Imágen 4)**, $k = 5$ **(Imagen 6)** y $k = 11$ **(Imagen 7).**

Un buen ajuste de la variable k puede mejorar los resultados del clasificador, todo dependerá de cómo estén distribuidos los datos. Usando valores grandes para k se reduce el efecto de ruido en la clasificación, pero puede generar más predicciones erróneas si existen clases parecidas

Para determinar la distancia entre los puntos, generalmente se usa la *distancia euclidiana.* Esta es especialmente útil cuando los datos de todas las variables son similares, como es el caso de este proyecto.

Si los datos contienen un alto número de variables, esto puede conducir a que datos parecidos aparezcan a distancias similares que datos que no lo son, y el clasificador falle, este problema se conoce como *Maldición de la Dimensionalidad (Curse of Dimensionality)*[3].

Para ilustrar este problema, los siguientes gráficos **(imagen 8)** muestran las distribuciones de todas las distancias por pares entre puntos distribuidos aleatoriamente dentro de un espacio cúbico de tamaño n. Donde n es el número de dimensiones.
A medida que n crece, todas las distancias se concentran dentro de un rango muy pequeño. Es decir, que todas las distancias entre puntos se acaban pareciendo, lo que perjudica seriamente el resultado.

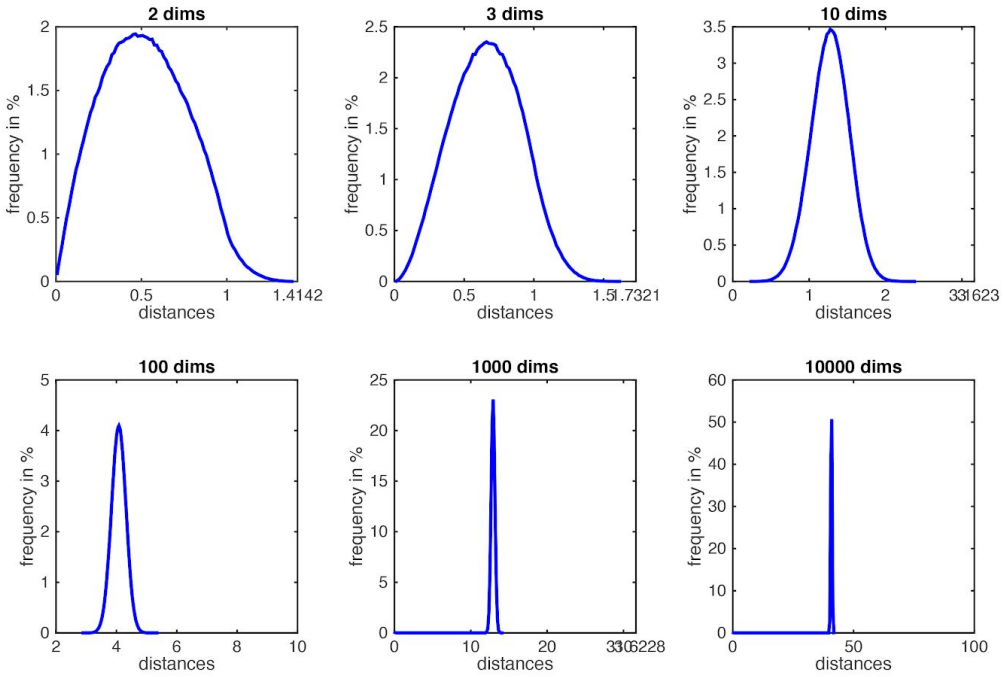


Imagen 8: Distribución de la frecuencia de distancias entre puntos dependiendo de la cantidad de dimensiones

Linear Discriminant Analysis (LDA)

El Análisis Discriminante Lineal (LDA) [4] es un método de clasificación supervisado que hace uso del teorema de Bayes [5] para estimar la probabilidad de que un nuevo dato pertenezca a cada uno de los posibles grupos.

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{P(B)}$$

Formula 1. Enunciado del teorema de Bayes

Adaptando el teorema de Bayes (**Fórmula 1**) se obtiene la siguiente (**Fórmula 2**) que se puede resumir como:

La probabilidad de que los datos x pertenezcan a la clase k es igual a la probabilidad de que en todos los datos de la clase k se encuentren los datos x, dividido por la probabilidad total de encontrar x en todo el conjunto de datos.

$$Pr(Y = k|X = x) = \frac{Pr(X = x|Y = k) \cdot Pr(Y = k)}{Pr(X = x)} \qquad Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}$$

Fórmula 2. Adaptación del Teorema de Bayes

Donde:

- $f_k(x)$ es la función de densidad para x en la clase k. Cómo están distribuidos los datos
- π_k es la probabilidad previa de pertenecer a la clase k. La proporción inicial de muestras de cada clase.

Hay que tener en cuenta que x en los datos de este proyecto está formado por un vector con todas las variables de cada observación. $x = (x_1, x_2, x_3, ...x_n)$

Para la predicción, se calcula la probabilidad de que cada nuevo x pertenezca a las distintas clases, y se asigna como predicción la clase para la cual esta probabilidad es mayor.

Support Vector Machine (SVM)

La *Máquina de Vector de Soporte* [6] es un modelo de aprendizaje ideado para el reconocimiento de patrones basado en el *principio de minimización del riesgo estructural* [7] .

Este algoritmo de clasificación funciona generando un campo con tantas dimensiones como variables tenga el conjunto. Seguidamente coloca en ese espacio los datos como coordenadas de los puntos y se les da la etiqueta del grupo al que pertenecen.

A continuación se busca los hiperplanos que mejor separan cada conjunto de las posibles etiquetas.

Para mostrar una representación visual simplificada (**Imagen 9**), se utilizan solamente dos variables ($X1, X2$), y un conjunto de etiquetas binario (blanco, negro). Estos datos se puede representar en un plano bidimensional usando los valores de las variables como coordenadas. Se se colocan en esos puntos y se busca la línea que mejor separa ambos.

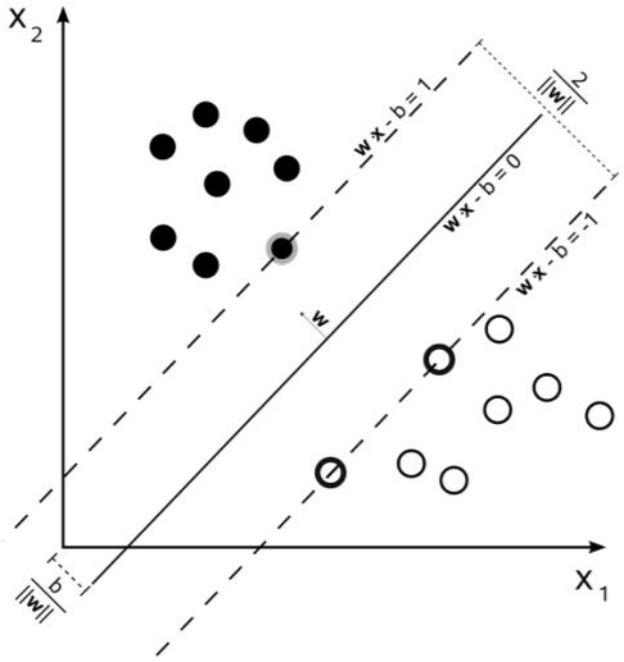


Imagen 9: Ilustración de la línea divisoria y margen creados.

En este sentido, encontrar la línea divisoria que separa con mayor margen las diferentes clases, se convierte en un problema de optimización.

Para la predicción, se colocan los nuevos datos en ese campo n-dimensional creado y se observa a qué sección del espacio n-dimensional pertenecen.

Se podría dar el caso de que con el número de dimensiones, los datos de entrenamiento (puntos en el espacio n-dimensional) no pudieran ser linealmente separados. Para solucionar este problema se utilizan funciones *Kernel* para modelos no-lineales y/o para que proyectar la información en un espacio dimensional mayor, añadiendo dimensiones extra que le ofrezcan otras posibles soluciones.

Para mostrar un ejemplo visual de este re-dimensionado, en la siguiente figura (**Imagen 10**) se pueden ver datos que no son linealmente separables en 2 dimensiones. Para solucionarlo, SVM añade una dimensión extra, transformando los datos y manteniendo la misma distancia entre clases. (**Imagen 11**).

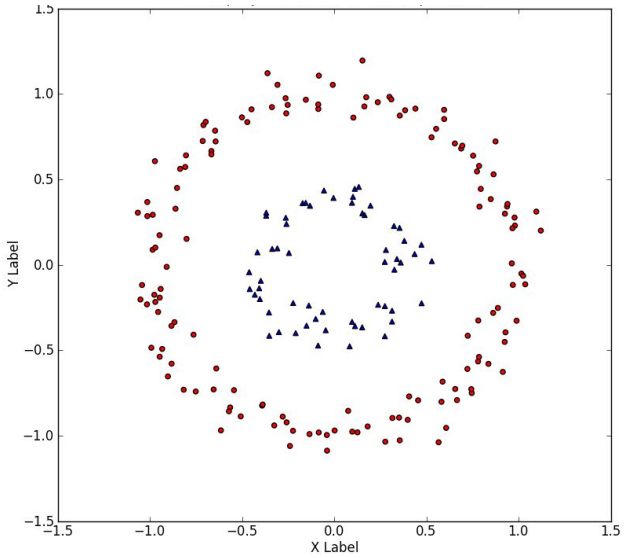


Imagen 10: Representación de datos no linealmente separables, en 2D.

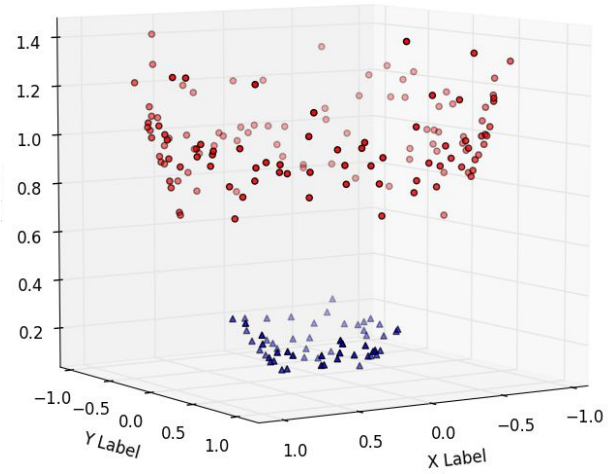


Imagen 11: Representación de los datos no linealmente separables con una dimensión extra.

De este modo se puede definir un hiperplano que crea esta división en una dimensión superior (**Imagen 12**)

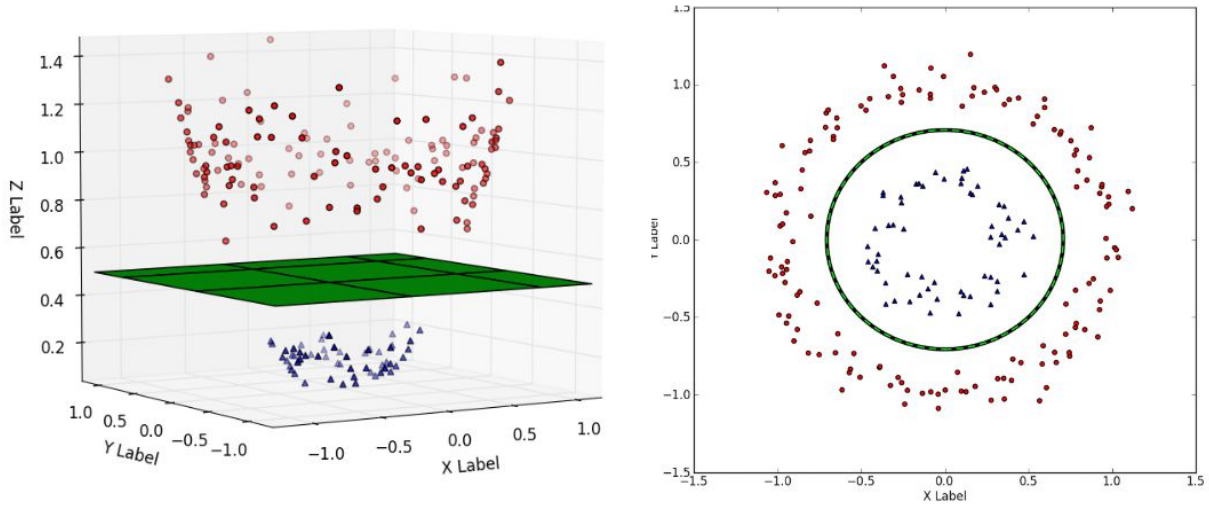


Imagen 12: Representación del hiperplano que divide las clases en 3D y en 2D.

Sin embargo la transformación de los datos puede ser muy costosa si el espacio es de dimensión muy alta por lo que es necesario asegurar lo suficiente el ajuste de los datos procesados para que contengan la información que mejor separa los distintos grupos, con el fin de no tener que hacer la transformación explícita de cada dato.

Las diferentes opciones de *Kernel* [8] que ofrece este modelo lo hace realmente adaptativo a los distintos tipos de datos y tamaños por lo que resulta realmente interesante comprobar cómo son los resultados de este clasificador con los datos de este proyecto.

También al tratar con una alta dimensionalidad, SVM se comporta mejor ya que mejora su aprendizaje con un mayor número de variables independientes y ha demostrado ser muy efectivo en clasificación de datos de texto [9], precisamente el tema central de este proyecto.

Para el caso particular de este proyecto, en que la mayoría de los datos de cada vector x será 0, se pueden optimizar el modelo, transformando los datos de entrenamiento en una matriz *Sparse* la qual solamente guarda los datos $\neq 0$ y su posición en la matriz, lo que supone un ahorro importante en tamaño de datos y tiempo de procesado.

La variable *Cost* (la constante de regularización *C* en la *formulación de Lagrange)* controla el coste que se utiliza para penalizar una clasificación errónea en la creación

de los hiperplanos que separan las distintas clases. Se podría decir que controla la "*dureza"* del margen entre clases.

Para crear estas líneas divisorias entre puntos de diferentes clases, se tiene en cuenta un margen de error ($Cost$) que decide con qué medida se penaliza un error de clasificación. Controlando y ajustando este coste se puede evitar la sobreespecialización del modelo (overfitting).

Finalmente señalar que el modelo generado con SVM, con los ajustes oportunos de sus variables y haciendo uso del *Kernel* que mejor se adapta al tipo de datos, proporciona un entrenamiento y predicciones muy eficientes [10].
De todos los tipos de *Kernels* existentes, en este proyecto investigaremos los siguientes.

Tipos de *Kernel*

- Lineal
- No Lineal
    - Polinómico
    - Base Radial Gaussiana *(RBF)*
    - Sigmoide

Lineal

Este tipo de *Kernel* busca una separación lineal (con una línea o hiperplano) que separe los datos en las n clases del modelo de datos (**Imagen 13**).

Para ajustarlo se puede controlar el valor de la variable *Cost,* presente en todos los modelos.

Como los datos de texto suelen ser linealmente separables, anticipamos que el *Kernel Lineal* será especialmente efectivo para la categorización de textos.[11]

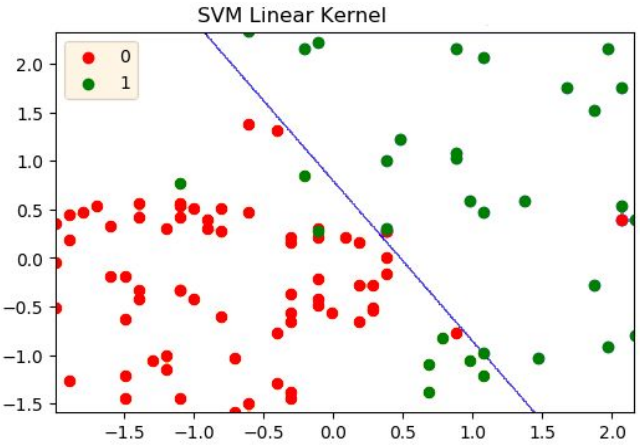


Imagen 13: Representación de la separación obtenida con SVM y *Kernel Lineal*

$$k\left(x_i, x_j\right) = \left(\gamma \cdot x_i \cdot x_j\right)$$

Formula 3: Ecuación del *Kernel Lineal*

No Lineal

Este tipo de *Kernel* busca una separación no lineal (conjunto de curvas) que ajuste mejor la separación entre clases. Para hacerlo se sirve de los datos que se encuentran en la cercanía de datos de otras clases, a los que considera como vectores de soporte.

Además de la variable *Cost,* para *Kernel no lineal* hace falta ajustar también la variable *gamma (γ),* para realizar una búsqueda de cuadrícula (*grid).*

Polinómico:

Generalmente usado en el procesado de imágenes, este *Kernel* busca el margen generando un polinomio de grado *d* que agrupe los distintos conjuntos de clases.

Ejemplo con dos clases (**Imagen 14**).

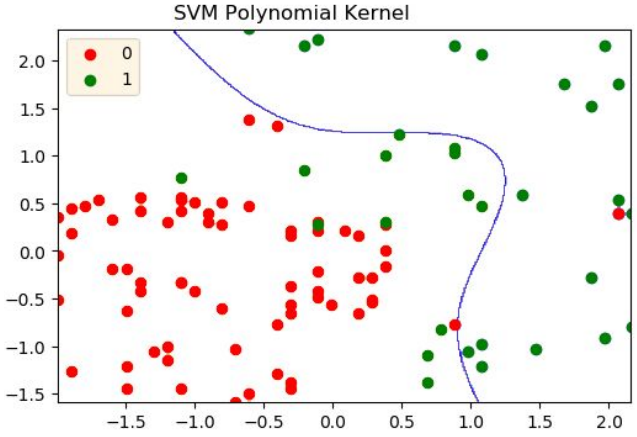


Imagen 14: Representación de la separación obtenida con SVM y *Kernel Polinómico*

$$k\left(x_i, x_j\right) = \left(\gamma \cdot x_i \cdot x_j + coef0\right)^d$$

Formula 4: Ecuación del *Kernel Polinómico*

Base Radial Gaussiana (Radial Base Function):

Es un *Kernel* de propósito general utilizado cuando no hay conocimiento previo sobre los datos.

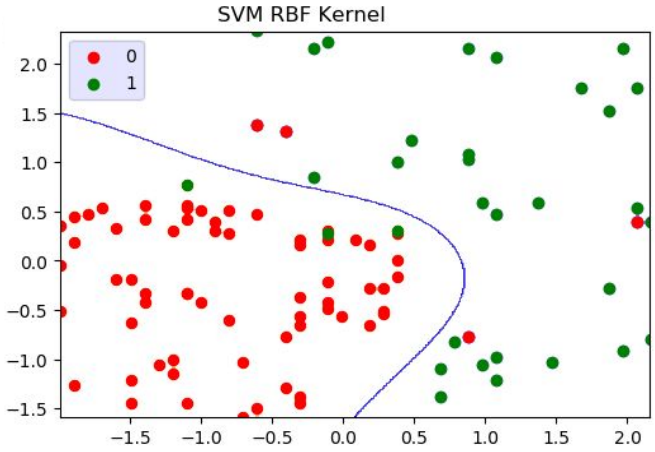


Imagen 15: Representación de la separación obtenida con SVM y *Kernel Base Radial Gaussiana*

$$k\left(x_i, x_j\right) = e^{\left(-\gamma\left\|x_i - x_j\right\|^2\right)}$$

Formula 5: Ecuación del *Kernel Base Radial Gaussiana*

Sigmoide:

Es otro tipo de *Kernel* de propósito general, popularizado en el uso de redes neuronales, que genera una forma *sigmoidea.*

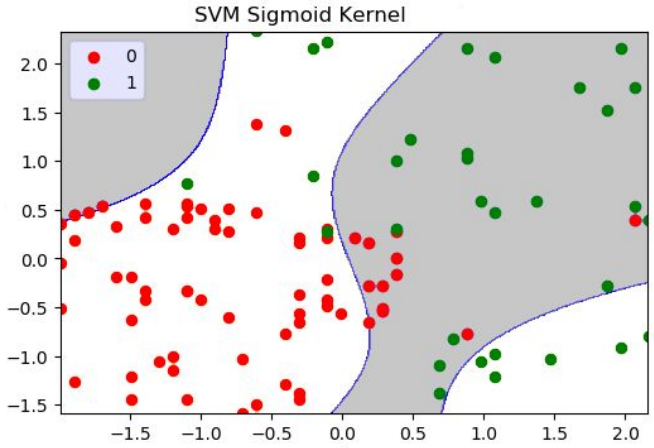


Imagen 16: Representación de la separación obtenida con SVM y *Kernel Sigmoide*

$$k\left(x_i, x_j\right) = \tanh\left(\gamma x_i x_j + coef0\right)$$

Formula 6: Ecuación del *Kernel Sigmoide*

5 - Pruebas y resultados

5.1 - Metodología

Para determinar el éxito de la clasificación utilizaremos la *Accuracy* obtenida en la *Matriz de confusión.*

Una matriz de confusión, es una matriz donde se comparan los datos reales con las previsiones. Permite una visualización del desempeño del clasificador.

	Pred					
True	PP	Cs	PSOE	ERC	jxcat	pod
PP	547	47	40	25	3	53
Cs	67	378	25	37	13	36
PSOE	50	28	285	13	0	54
ERC	23	17	1	344	77	53
jxcat	13	10	6	116	436	8
pod	72	33	19	41	5	466

Imágen 17: Matriz de confusión

En este ejemplo (**Imagen 17**), como filas, a la izquierda aparecen los datos reales, y como columnas, los datos predichos.

Contra mayor sean los valores que se encuentran en la diagonal, mejor está funcionando la clasificación

La *Accuracy* es el sumatorio de la diagonal, dividido por el conjunto total de datos de prueba. Dicho de otro modo: es la proporción de Acierto sobre el total.

> En este proyecto hemos traducido *Accuracy* como **Exactitud***,* para evitar confundir el concepto con el de *Precisión.*

5.2 - Implementación

En la ejecución de este proyecto hemos usado dos lenguajes de programación.

Python, por su conocimiento previo, facilidad de tratamiento de datos y optimización.

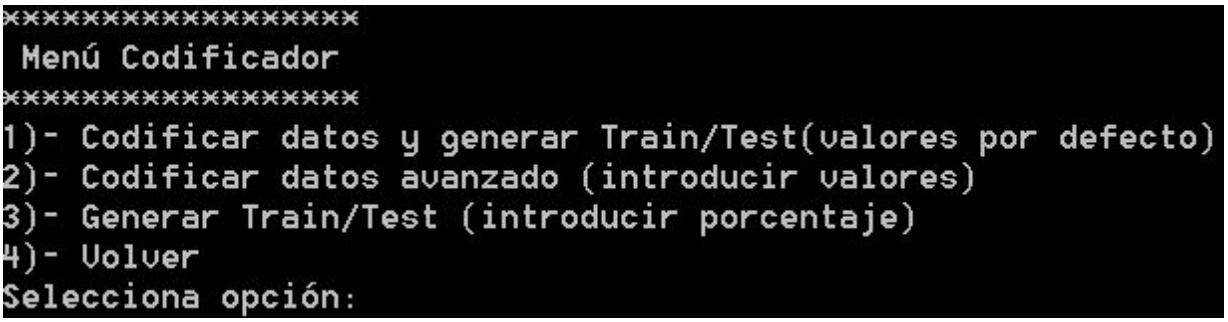
Y *R* para ejecutar los modelos de predicción y tratamiento de *dataframes (*usando *scripts).* Este lenguaje se adapta perfectamente a las necesidades de este proyecto dado su enfoque al tratamiento de datos analítico, y también por la posibilidad de cargar diferentes bibliotecas con modelos de aprendizaje automático y funcionalidades de cálculo y graficación.

En este proyecto estos lenguajes se han usado para:

- Python
    - Extracción de datos
    - Tratamiento y gestión de las palabras
        - Librería usada: *nltk.stem.porter* para la *stem / lematización* de las palabras.
    - Creación de los vectores de datos finales
        - Librería usada:
    - Llamada a las funciones en *R*
        - Librería usada: *subprocess* para hacer las llamadas a los *scripts* en *R,* y pasarles las variables necesarias.

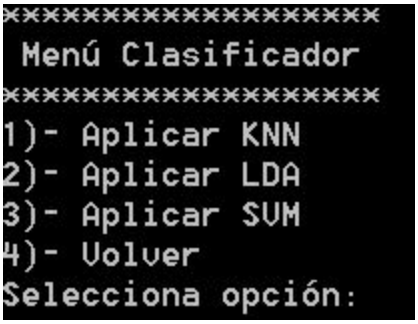
- R
    - Creación de los conjuntos *Train / Test*
    - Creación de gráficos
    - Creación y aplicación del modelo KNN
    - Creación y aplicación del modelo LDA
    - Creación y aplicación del modelo SVM
        - Librerías usadas: *caret, lattice, e1071, Matrix, ggplot2*

El programa creado con *Python,* funciona por consola ofreciendo: un menú para codificar los datos (**Imágen 18**)



Imágen 18: Menú codificador

Y también un menú para ejecutar los distintos modelos clasificadores, y los ajustes a los distintos valores de cada clasificador.(**Imágen 19**)



Imágen 19: Menú clasificador.

Cada modelo tiene distintos parámetros que se pueden ajustar. Sin embargo se recomienda usar el que aparece por pantalla “por defecto”

Finalmente, después de la clasificación se muestra por pantalla la matriz de confusión, y la exactitud (Accuracy), con el porcentaje de Error (que es 1-Accuracy). También se muestran el valor de las variables y el tiempo que ha tardado en crear el modelo y generar la predicción.

Ejemplo ajustando el tipo de Kernel de SVM a Lineal y con Cost=0.1 y Tolerance =0.1 (**Imágen 20**)

Imágen 20: Ejecución del modelo SVM con Kernel Lineal

5.3 - Distribución de datos

El tratamiento de datos en este proyecto consiste en la extracción de la información más relevante. Dado que tratamos con datos de texto, hemos decidido que será conveniente para nuestro modelo conocer cuál es la proporción de uso de cada palabra / pareja de palabras. Este porcentaje nos servirá para asignar el conjunto de datos que consideremos de mayor importancia.

Hay que tener en cuenta que el porcentaje de uso de palabra, se contabiliza dentro de su clase, en este caso, dentro de cada partido político. Es decir que la misma palabra probablemente tendrá porcentajes distintos de uso dependiendo del partido político que estemos analizando.

Para ello utilizamos las variables $MIN, MAX, MINP, MAXP$ que hacen referencia al rango de uso. (Para palabras y para parejas de palabras, respectivamente)

Si una palabra es usada un porcentaje de veces dentro de ese rango se considerará un buen candidato.
Para finalizar, si también ha sido usada por, como máximo, el número de partidos establecido por parámetro (cw, cp), esta palabra pasará a formar parte de las variables independientes.

Para poder definir apropiadamente estos límites primero debemos conocer cual es la distribución de los datos.

Observamos las 5 palabras más usadas y sus porcentajes (**Tabla 1**). Este porcentaje se ha calculado como: la cantidad de veces que ha aparecido la palabra / pareja (dentro de los *tweets* del partido), dividido por el número de *tweets* de ese partido.

> Recordemos que estas son los lemas de las palabras originales. Esto genera cambios en cómo vemos la palabra final. Algunas palabras quedan cortadas: Ejemplo "*Junqueras"* como apellido, lo reconoce como la palabra *junquera,* y deja esta como lema.

Como se puede ver en la **Tabla 1**, la mayoría de estas palabras (o parejas) son meros conectores, artículos o preposiciones. Realmente no parecen ofrecer información discriminativa entre grupos.

PP		C's		PSOE	
palabra	**porcentaje**	**palabra**	**porcentaje**	**palabra**	**porcentaje**
que	68.19%	que	91.19%	que	48.62%
del	36.50%	para	38.30%	del	27.36%
con	33.79%	del	36.48%	por	23.01%
para	31.86%	por	34.10%	con	22.29%
por	26.26%	con	29.26%	para	21.48%
pareja	**porcentaje**	**pareja**	**porcentaje**	**pareja**	**porcentaje**
de la	25.39%	de la	28.16%	de la	21,33%
a la	15.24%	a la	19.69%	a la	14,38%
en la	11.18%	a lo	15.36%	en el	8.54%
en el	10.99%	en el	15.07%	en la	7.15%
de lo	10.43%	en la	14.37%	de lo	7.17%

ERC		Jxcat		Podemos	
palabra	**porcentaje**	**palabra**	**porcentaje**	**palabra**	**porcentaje**
que	58.26%	que	68.39%	del	33.88%
per	28.00%	per	43.40%	para	33.63%
del	27.34%	del	38.22%	por	30.70%
junquera	27.26%	artadi	24.75%	con	28.32%
una	18.07%	elsa	24.68%	una	25.31%
pareja	**porcentaje**	**pareja**	**porcentaje**	**pareja**	**porcentaje**
de la	17.33%	elsa artadi	24.68%	de la	27.51%
oriol junquera	15.36%	de la	23.96%	a la	18.53%
rovira vergé	15.01%	a la	14.38%	en la	11.11%
a la	12.17%	torra i	13.67%	en el	10.77%
marta rovira	10.70%	carl puigdemont	11.55%	a lo	10.48%

Tabla 1: Las 5 palabras y parejas más usadas por cada partido, y su porcentaje de uso.

Ahora que conocemos los máximos (% de uso) queremos conocer cómo es esta distribución.

> Recordemos que para calcular la % de uso de palabra, se cuenta cuántas veces se ha usado por un partido y se divide por el número de *tweets* de ese partido.

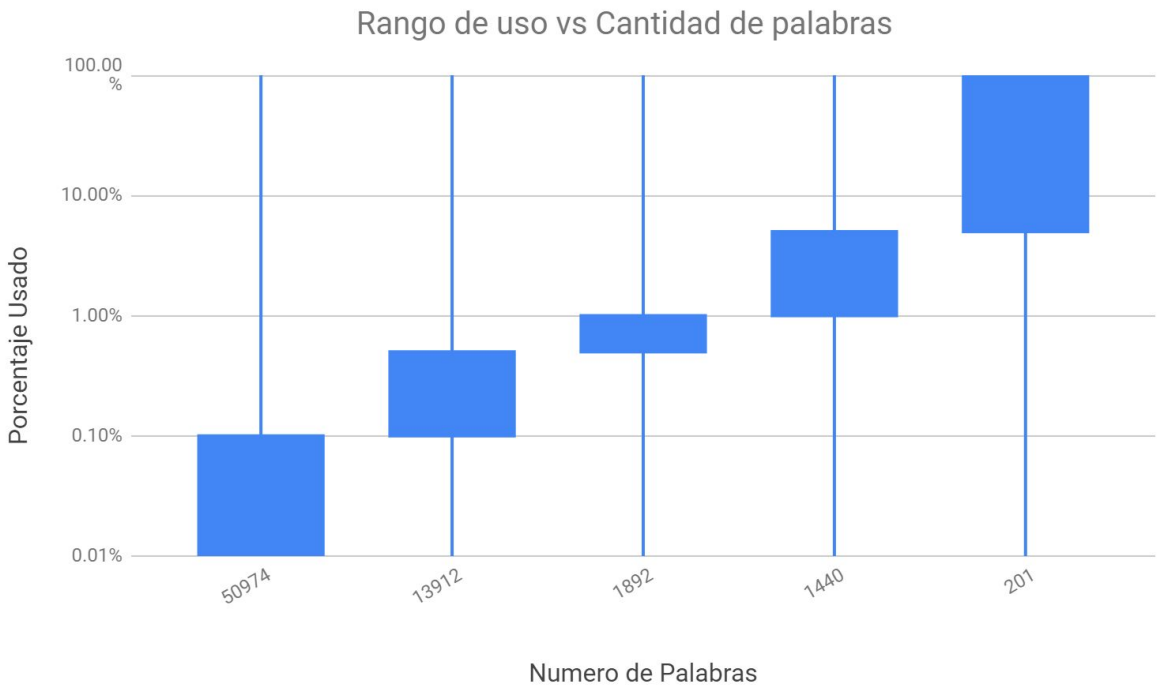
Como las distribuciones de los diferentes partidos son muy parecidas, nos centramos solo en una para ver en qué rangos está el grueso de los datos.

Imágen 21: Distribución de la cantidad de palabras y su porcentaje de uso (partido ERC)

La mayoría de palabras tienen un porcentaje de uso de menos del 5%.

Ahora queremos conocer cuántas palabras obtendremos de **todos los partidos**, cuando introducimos rangos con las variables *MIN, MAX*

> Recordemos que *MIN, MAX* son los límites Mínimo y Máximo que utilizamos para extraer palabras. Las palabras extraídas habrán sido usadas un % de veces dentro de ese rango (*MIN - MAX)*.



Imágen 22 : Rango respecto la cantidad de palabras que han sido usadas dentro de ese porcentaje

Se puede ver (**Imágen 22**) que la mayoría de las palabras (~60.000) se usan entre el *0.01% y el 0.5%* (los dos primeros bloques de la izquierda)

Por otra parte, en el bloque de la derecha, solo se encuentran ~200, palabras entre el *5%* y el *100%* de uso.

Probablemente no nos interesan ninguno de estos rangos.

A continuación observamos cómo se comportan las parejas de palabras y analizaremos ambos conjuntos(palabras y parejas) y sus rangos más útiles.

Imágen 23: Distribución de la cantidad de palabras de palabras y su porcentaje de uso (partido C's)

Como en las palabras, la mayoría de parejas de palabras tienen un porcentaje de uso de menos del **5%.**
Aunque la distribución es similar al de las palabras, si usamos mismos rangos habrá menos cantidad de parejas que de palabras útiles.

Para hacer pruebas con la misma cantidad de variables con palabras y parejas necesitaremos mínimos y máximos distintos a los usados anteriormente (*MIN, MAX)* para extraer los datos más relevantes.

Ahora queremos conocer cuántas palabras obtendremos de **todos los partidos**, cuando introducimos rangos con las variables *MINP, MAXP.*

> Recordemos que *MINP, MAXP* son los límites Mínimo y Máximo que utilizamos para extraer **parejas de palabras**.

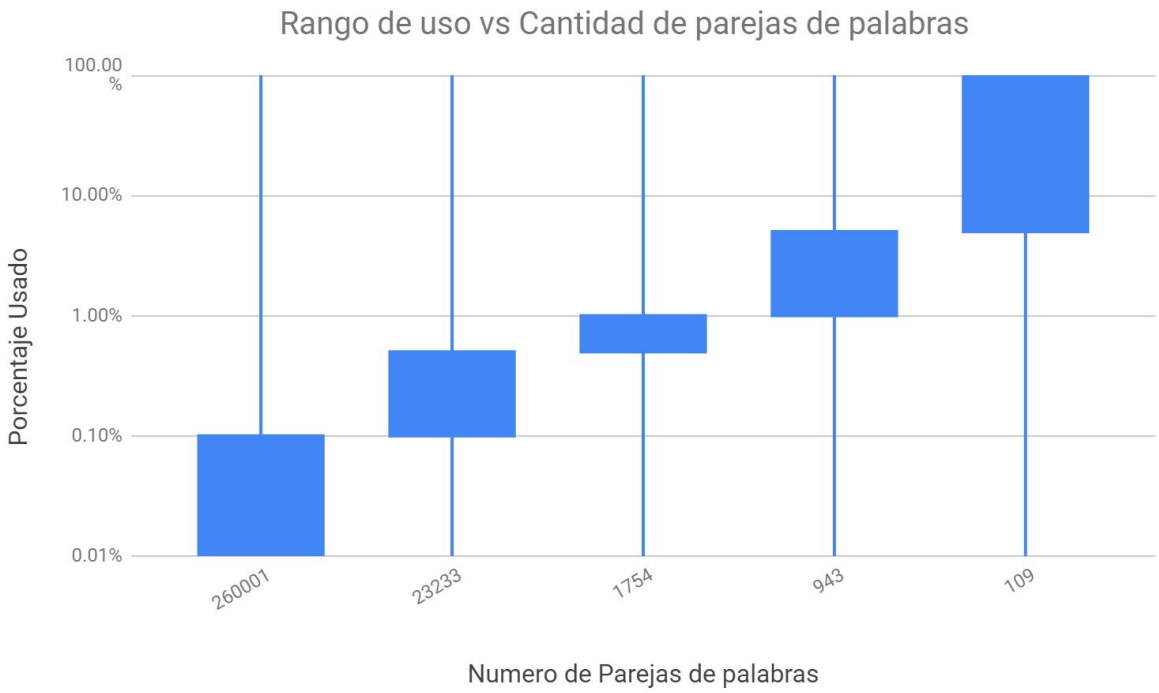


Imagen 24: Rango respecto la cantidad de parejas de palabras que han sido usadas dentro de ese porcentaje

Se puede ver (**Imágen 24**) que la mayoría de las palabras (~50.000) se usan entre el *0.01% y el 0.5%* (los dos primeros bloques de la izquierda)

Por otra parte, en el bloque de la derecha, solo se encuentran ~100, parejas (entre el *5%* y el *100%* de uso).

Probablemente no nos interesan ninguno de estos rangos.

Analizando los rangos :

- Entre *0.01%* y *0.1%*: Palabras con tan poco uso que implica que han aparecido en muy pocos *tweets* y supondremos que no aportan mucha información.
- Entre *0.5%* y *5%*: Palabras usadas suficientes veces como para aparecer en bastantes *tweets*, pero no demasiadas como para ser genéricas.
- Entre *5%* y *100%*: Palabras tan corrientes que son usadas de forma general y probablemente no ayudan a separar grupos

En la fase de pruebas queremos averiguar si el modelo funciona mejor con **solo palabras**, **solo parejas**, o usando **ambas**.

Además nos interesa ver cómo evoluciona el éxito en la predicción con distintos valores para *cw, cp* .

Ajustamos Mínimos y Máximos para tener: **Tres conjuntos** de datos **solo con palabra**s. Y otros tres **solo con parejas de palabras**.

Para mayor facilidad les asignaremos:

- **A** al conjunto con ~500 variables independientes.
- **B** al conjunto con ~1500.
- **C** al conjunto con ~3500.

Es necesario aclarar que se crearan los conjuntos **A,B,C** utilizando sólo palabras, y luego se crearán otros **A,B,C** utilizando sólo parejas. (Manteniendo las cantidades antes mencionadas)

Nos referiremos a ellos por el mismo nombre (**A,B,C**) para ahorrarnos llamarlos por su cantidad de variables.
Expondremos los datos por separado para evitar confusión.

Para crearlos, los tres grupos de palabras mantienen el mismo valor para el Máximo (MAX), y se utilizan Mínimos cada vez más pequeños, para que el rango abarque cada vez más cantidad de datos.

Para los grupos de parejas se hace lo mismo, pero con otros valores MINP,MAXP.

Hemos realizado algunas pruebas para encontrar Mínimos y máximos relevantes antes de llevar a cabo las que presentamos aquí, con estos valores hemos creado los grupos **A,B,C**

Grupo A		**Grupo B**		**Grupo C**	
Palabras	Min = 1% Max = 5%	**Palabras**	Min = 0.5% Max = 5%	**Palabras**	Min = 0.25% Max = 5%
Parejas	Minp = 1% Maxp = 50%	**Parejas**	Minp = 0.5% Maxp = 50%	**Parejas**	Minp = 0.3% Maxp = 50%
Variables	~500	**Variables**	~1500	**Variables**	~3500

Tabla 3: Configuración de los Mínimos y Máximos que se han utilizado para crear los grupos **A,B,C**.

Debido a esto, el grupo más pequeño (grupo **A**), estará únicamente formado, únicamente, por palabras mucho más corrientes que el resto de grupos.

Por el contrario, grupos con más datos (como el **C**) incluyen además, palabras con menor porcentaje de uso, es decir menos corrientes, lo que pueden suponer una ventaja.

Veamos una pequeña muestra de qué palabras y parejas hay en cada uno de los grupos:

Grupo A		Grupo B		Grupo C	
Palabras	**Parejas**	**Palabras**	**Parejas**	**Palabras**	**Parejas**
español	el dia	articulo	lo independentista	congreso	union y
cataluña	el apoyo	proceso	de oportunidad	familia	sociedad catalana
partido	con un	fugado	vez de	memoria	constitución española
gobierno	defendiendo la	populismo	sede de	derecho	legalidad y

Tabla 4: Algunas de las palabras y parejas que se han utilizado para crear los grupos **A,B,C**.

Suponemos que una ajuste bien equilibrado de estos porcentajes puede servirnos para mejorar el resultado del clasificador. Lo comprobaremos en el apartado de pruebas.

5.4 - Clasificación

Pruebas con LDA

Para conocer en qué medida es preciso este modelo primero debemos probarlo con diferentes transformaciones de los datos para determinar con qué, valores en la codificación de datos, se producen mejores resultados.

Para LDA es importante evitar variables con desviación estándar cero, y también es conveniente eliminar las variables que tengan alto porcentaje de correlación (>90%).

Pruebas con **A**,**B**,**C** formados sólo con palabras.(**Imagen 25**).

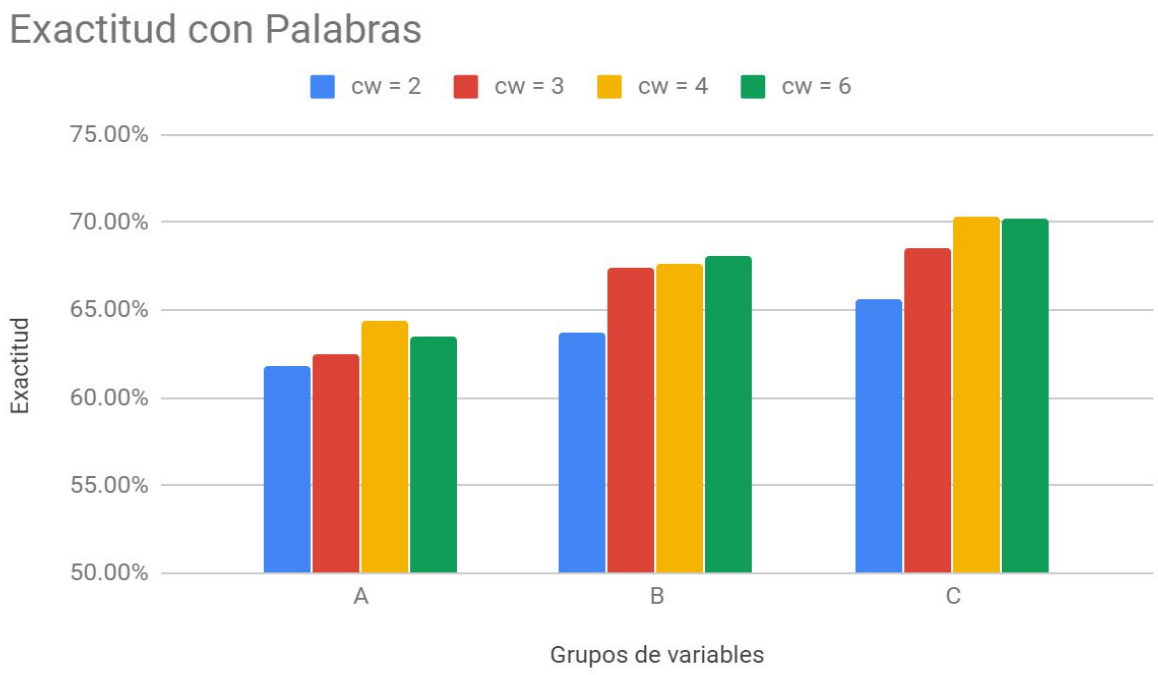


Imagen 25: Exactitud con LDA usando solo palabras,
con distintas cantidades de variables (A,B,C) y valores para *cw*.

Pruebas con **A**,**B**,**C** formados sólo con parejas (**Imagen 26**).

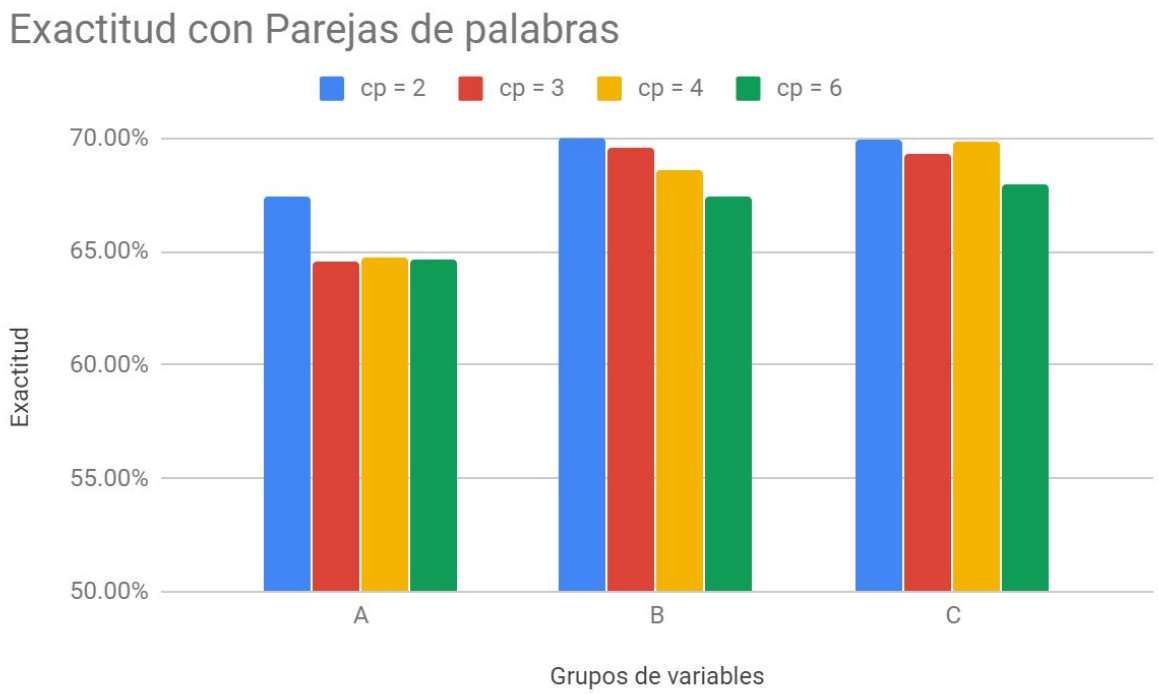


Imagen 26: Exactitud con LDA usando solo parejas de palabras, con distintas cantidades de variables (A,B,C) y valores para *cp*.

Observando estos gráficos podemos afirmar que, en general, el uso únicamente de **parejas de palabras** obtiene resultados parecidos a los obtenidos con solo palabras, pero con usando una cantidad menor de variables.

Variables (grupo)	Uso	Exactitud	Ajuste
B	Palabras	71.25%	cw = 6
C	Parejas de palabras	68.05%	cp = 4

Tabla 5: Con LDA, mejores resultados de exactitud para Palabras y para Parejas de palabras.

Para comprobar si podemos mejorar los resultados, crearemos un conjunto de datos que contenga los dos mejores conjuntos (**Tabla 5**) y sus respectivos ajustes (*cw,cp)*

El resultado del clasificador y detalles del conjunto de datos (**Imágen 27**)

Variables	4534
Exactitud	72.42%
Uso	ambos
cw - cp	4 - 2
MIN	0.25
MAX	5
MINP	0.5
MAXP	50

	Pred					
True	PP	Cs	PSOE	ERC	jxcat	pod
PP	563	38	62	6	2	44
Cs	75	348	42	26	13	41
PSOE	67	15	370	8	6	57
ERC	24	5	18	365	62	38
jxcat	10	4	18	119	419	12
pod	60	18	52	23	4	479

Imagen 27: Detalles del modelo, la exactitud obtenida con LDA y la matriz de confusión generada

Este ajuste de los datos proporciona una exactitud del 72.42%, superior a las obtenidas por los por los conjuntos que lo forman (**Tabla 3**) 71.25% y 68.05% respectivamente.

Después de hacer otras pruebas, confirmamos que uno de las mejores configuración para los datos en LDA, es la mostrada en la **Imágen 27** consiguiendo una **exactitud del 72.42%**,

En la matriz de confusión vemos que los partidos que peor ha clasificado son *C's* y *ERC*. (Sus valores en la diagonal son menores).

A continuación se muestra un extracto de los *tweets* clasificados y en qué grupo los ha colocado. Mostramos más de estos dos grupos peor clasificados (*C's* y *ERC*). (**Tabla 6**)

Marcamos en rojo los casos de clasificación errónea.

Real	Predicción	Tweet
PP	PSOE	The Men in Black Todo muy Partido Socialista OBRERO Español
PP	PP	Aqui podéis ver los cachorros de @QuimTorraiPla preparando el ataque a los @mossos Esas bolsas con pintura contra la policía es la munición que ha suministrado Torra al radicalismo con sus insultos y descalificaciones diarias contra el orden y el estado de derecho
Cs	Cs	¡Buenos días! A partir de las 8 30h os espero en @LasMananas_rne en @rne donde me entrevista @MenendezRNE
Cs	PP	Un placer volver a charlar con @MichelBarnier que está haciendo un duro trabajo como negociador de la UE ante el Brexit It has been a pleasure to talk again with @MichelBarnier on Brexit He is doing a great job leading the negotiations
Cs	pod	Muy atentos a la evolución del incendio en Llutxent Mi agradecimiento a los cuerpos de emergencia @UMEgob y voluntarios que trabajan duro desde hace días para sofocar las llamas
Cs	PSOE	Ha sido un orgullo presentar al gran equipo que liderará @Cs_Andalucia en las próximas autonómicas Personas capaces que saben lo que es trabajar fuera de la política y con ilusión por cambiar las cosas Casi 40 años de gobierno socialista son demasiados
PSOE	Cs	Un dosier de los Mossos censó a contrarios a la independencia Luis Rendueles y Vanesa Lozano
PSOE	PSOE	El internacionalismo está en el ADN socialista así lo voy a recordar ante el Consejo de la Internacional Socialista
ERC	jxcat	2 Recordem a tothom que pot anar a votar a qualsevol punt de votació! I que ha de vetllar els punts de votació un cop hagin votat!
ERC	ERC	Segueixen sota la cuirassa del TC Nosaltres com sempre seguim endavant complint el mandat democràtic
ERC	pod	@marcbataller @otcortes @Esquerra_ERC @junqueras @LlSalvado @jandreud Pep és membre de l'executiva com membre nat pq és President regional
jxcat	PSOE	Enhorabuena presidente Una distinción que reconoce los esfuerzos por la paz Con coraje y diálogo se solucionan conflictos @JuanManSantosCarles Puigdemont Manuel Santos @JuanManSantosEsta honrosa distinción no es para mí es para todas las víctimas del conflicto Juntos ganaremos el premio más importante de todos: LA PAZ
jxcat	jxcat	“Do you want #Catalonia to become an independent state in the form of a Republic #1Oct
pod	pod	Leed esto Seréis más felices y mejores personas después de hacerlo
pod	Cs	No son ciudadanos con banderas son fascistas PP y Cs deben dejar de compartir manifestaciones con ellos Los demócratas somos antifascistas

Tabla 6: Muestra de *Tweets,* el grupo al que realmente pertenecen y la clasificación que se les ha otorgado.

Pruebas con KNN

Las fase de pruebas para este modelo incluye conocer la mejor K para ajustar la mejor clasificación, evitando el sobreajuste del modelo.
Tambíen es importante recordar que este modelo es probablemente el menos óptimo para el tipo de datos con que tratamos en este proyecto por lo que sus porcentajes de exactitud tenderán a ser menores, y el tiempo computacional aumentará exponencialmente respecto al número de variables, por lo que es prudente que la dimensionalidad del espacio sea lo más baja posible (*Curse of Dimensionality)*.

Para encontrar el mejor valor de *k* , aplicamos distintos valores para esta variable y hacemos pruebas con validación cruzada en 5 porciones(*Cross Validation)*

Esto significa que el conjunto de datos se divide en 5 porciones,
(llamemoslas p1, p2, p3, p4, p5).

- En la primera iteración p1 hace de Test, y el resto (p2- p5) hacen de Train.
- En la segunda iteración p2 hace de Test y el resto (p1,p3,p4,p5) hacen de Train.

Y así sucesivamente. Llevando a cabo este bucle una vez por cada valor de *k (1,3,5,7)*

Finalmente, se calcula la media de los resultados obtenidos en cada bucle y se muestra cual es el valor de *k* que ofrece mejores resultados.

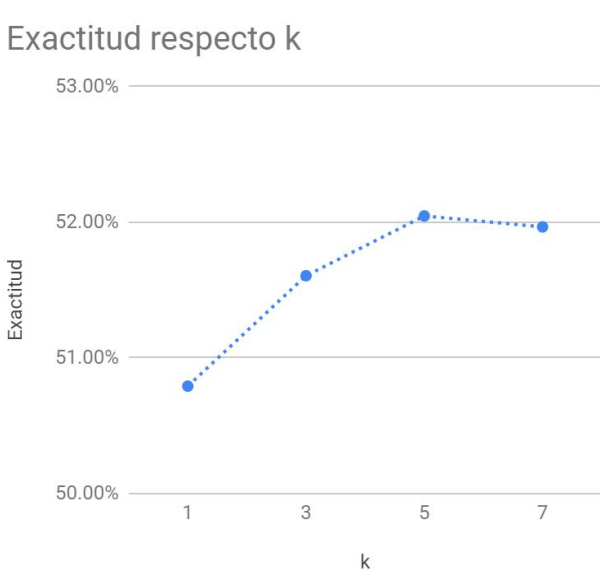
```
k-Nearest Neighbors

13636 samples
1027 predictor
6 classes:'PP','Cs','PSOE','ERC','jxcat','pod'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 10909, 10910,
10908, 10907, 10910
Resampling results across tuning parameters:

  k  Accuracy   Kappa
  1  0.5079184  0.4070347
  3  0.5160591  0.4160432
  5  0.5204616  0.4208462
  7  0.5196574  0.4194948

Accuracy was used to select the optimal
model using the largest value.
The final value used for the model was k = 5.
```



Imágen 28: Exactitud con KNN usando distintos valores para *k* con validación cruzada en 5 porciones (CrossVal 5 fold).

El mejor valor encontrado para *k* es 5 (Imágen 28)

Ahora observamos, igual que antes, cómo varía la exactitud con distintas cantidades de variables **A,B,C** (~500, ~1500, ~3500) respectivamente, usando únicamente palabras o parejas. También variamos los valores para *cw, cp*.

Pruebas con **A**,**B**,**C** formados sólo con palabras (**Imagen 29**).

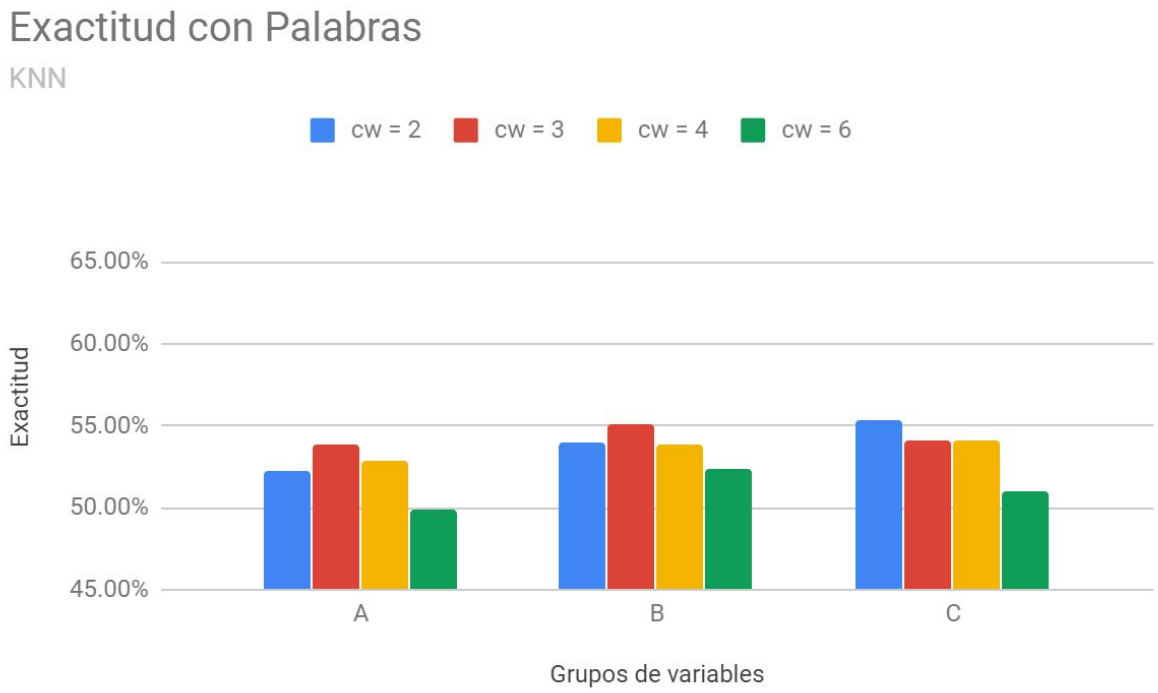


Imagen 29: Exactitud con KNN, k= 5,, usando solo palabras, con distintas cantidades de variables (A,B,C) y valores para cw

Pruebas con **A**,**B**,**C** formados sólo con parejas (**Imagen 30**).

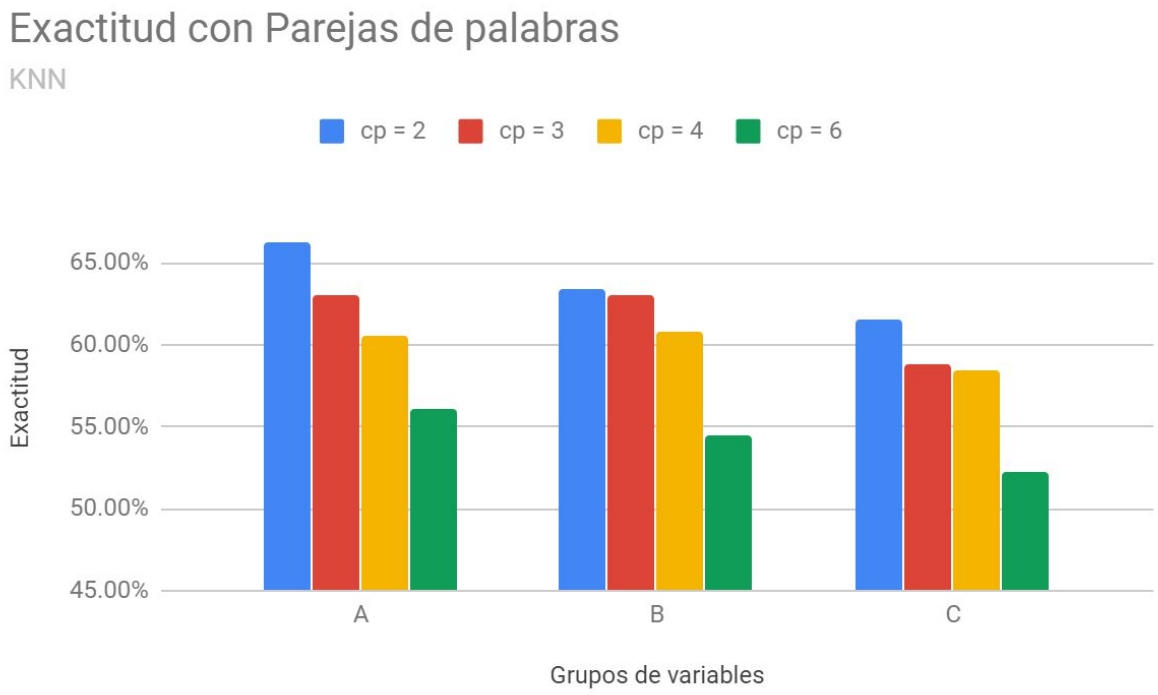


Imagen 30: Exactitud con KNN, k= 5, usando solo parejas de palabras, con distintas cantidades de variables (A,B,C) y valores para cp

En estos gráficos (**Imagen 30**) se observa cómo el número de variables es realmente decisivo a la hora de generar mejores predicciones. Como ya anticipamos, al tratar con alta dimensionalidad los resultados cada vez son peores, por lo que el mejor conjunto de datos para este modelo será uno con pocas variables

Se ha obtenido una **exactitud del 66.22%** con un conjunto relativamente pequeño de parejas de palabras. Ante la complejidad del problema y el modelo usado, consideramos que es un resultado altamente satisfactorio.

Pruebas con SVM

Las fase de pruebas para este modelo incluye conocer que tipo de *Kernel* es el que resulta más apropiado para el tipo de datos, además de ajustar otros posibles parámetros una vez determinado el *Kernel.*

Para las pruebas de *Kernel* y variables del tipo de *Kernel* utilizaremos dos conjuntos de datos, que usarán palabras y parejas a la vez:

Un conjunto **mediano**, formado con los grupos **A** (de solo palabras), y **A** (de solo parejas). El conjunto mediano tendrá ~1000 variables. (cada grupo **A** está formado por ~500 var.)

Y, un conjunto **grande,** formado con los grupos **B** (de solo palabras), y **B** (de solo parejas). El conjunto mediano tendrá ~3000 variables. (cada grupo **B** está formado por ~1500 var.)

Pruebas con tipos de *Kernel* (usando los valores por defecto): *Lineal*, *Polinómico, Sigmoide, Base Radial Gaussiana*

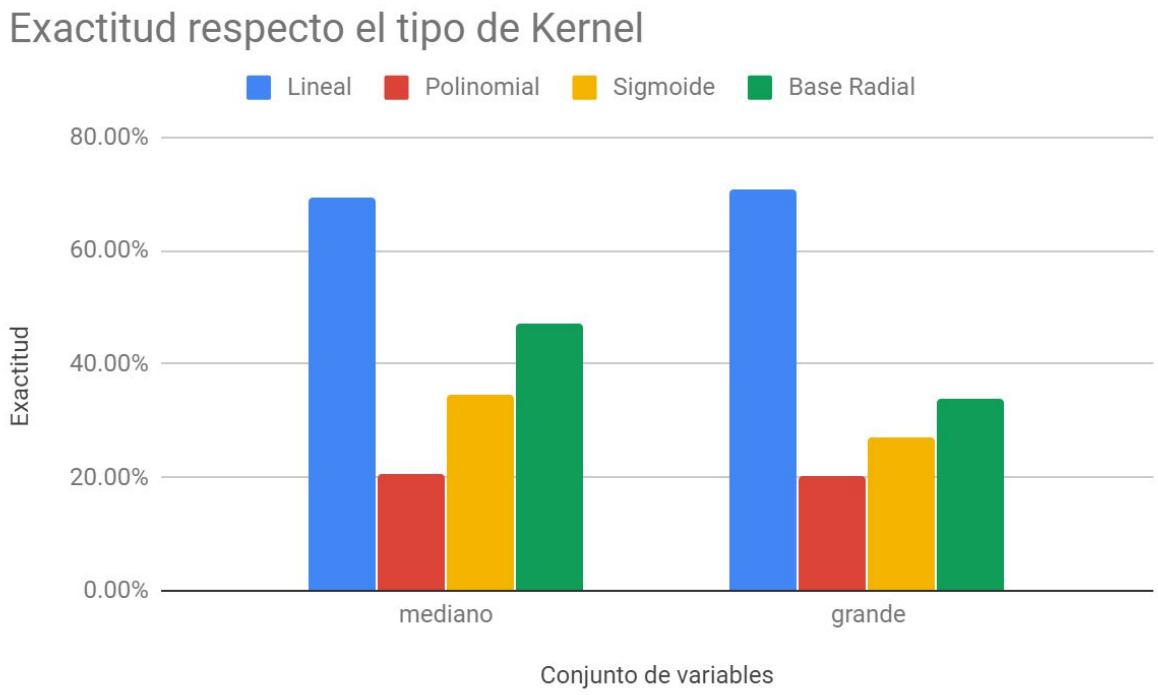


Imagen 31: Exactitud obtenida con SVM usando diferentes Kernels y conjuntos de variables

Observando los resultados (**Imagen 31**) resulta evidente que el mejor tipo de *Kernel,* para el tipo de datos de este proyecto, es *Kernel Lineal.* Sin embargo también probaremos a ajustar uno de los *Kernels no lineales,* para ver si podemos mejorar los resultados del *lineal.*
Escogemos el mejor *Kernel no lineal* observado en el gráfico*: Base Radial Gaussiana*.

Haremos pruebas ajustando los distintos valores de cada tipo de *Kernel* para conocer cómo pueden mejorar la predicción.

Cost - Este valor (la constante de regularización *C* en la *formulación de Lagrange)* controla el coste que se utiliza para penalizar una clasificación errónea en la creación de los hiperplanos que separan las distintas clases. Se podría decir que controla la "*dureza"* del margen entre clases.

Tolerance - Este valor es la tolerancia para los criterios de parada. Le dice al modelo que deje de buscar cuando el resultado no mejora, al menos en la cantidad *tolerance*, durante dos iteraciones consecutivas. Se considera que se alcanza la convergencia una vez que esté lo suficientemente cerca y se detiene el entrenamiento.

Estas variables afectan a todos los modelos, por lo que haremos pruebas con ambos *(Lineal* y *Base Radial Gaussiana*).

Pruebas con SVM Kernel Lineal

Con la variable *Cost,* probamos distintos valores para mejorar el modelo.

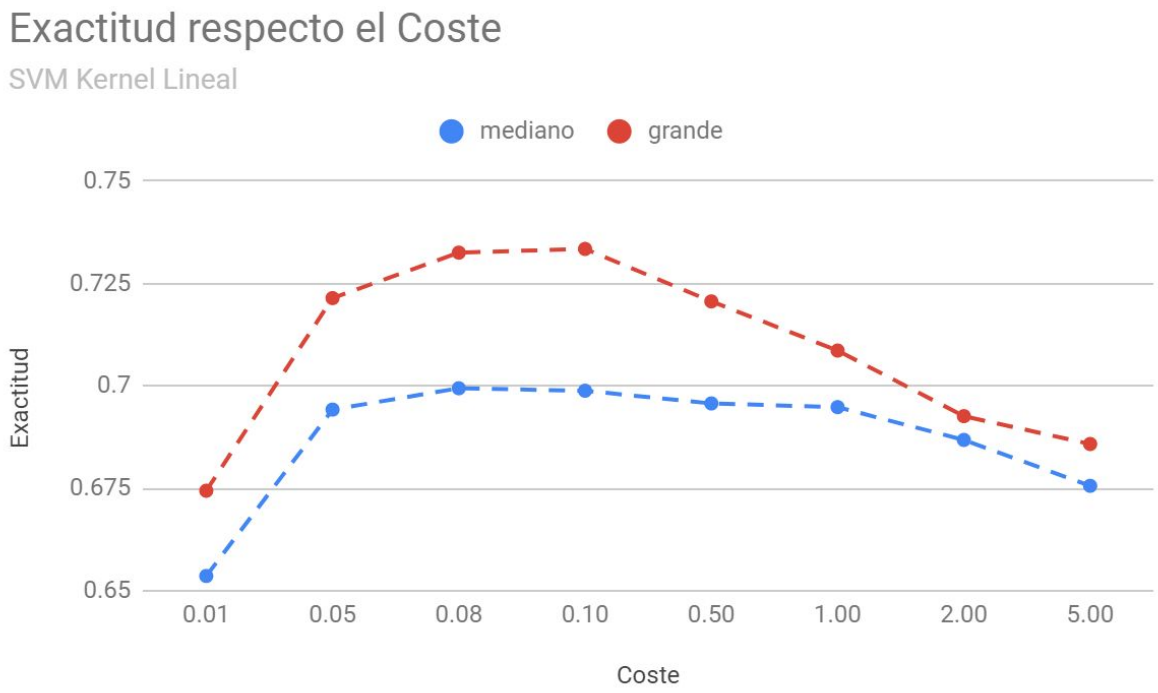


Imagen 32: Exactitud obtenida con SVM usando Kernel *Lineal* con los dos conjuntos de variables y distintos valores para *Cost.*

Los mejores valores para *Cost* parecen estar entre 0.08 y 0.1
Para nuestras pruebas usaremos:
Cost = 0.1

Pruebas con la variable *tolerance.* Este valor dependerá del *Kernel* y modelo que esté ajustando. Como no existe una tolerancia universal hacemos pruebas con diferentes valores para ver si con un buen ajuste podemos mejorar los resultados. Por defecto *tolerance = 0.001*

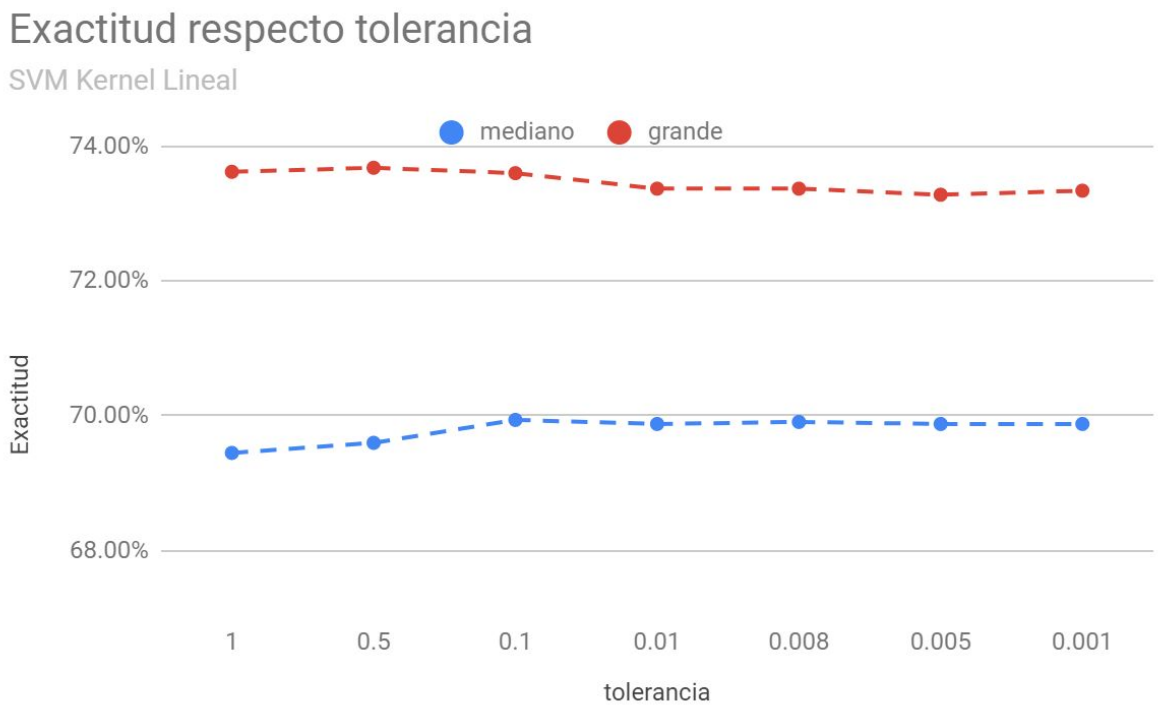


Imagen 33: Exactitud obtenida con SVM usando Kernel *Lineal, Cost = 0.1* con los dos conjuntos de variables y distintos valores para *Tolerancia*

La mejor tolerancia encontrada es 0.1

Ahora, para conocer qué conjuntos de datos funcionan mejor hacemos pruebas con distintas cantidades de variables **A, B, C**, usando únicamente palabras o parejas. También variamos los valores para *cw, cp*.

Pruebas con **A**,**B**,**C** formados sólo con palabras (**Imagen 34**).

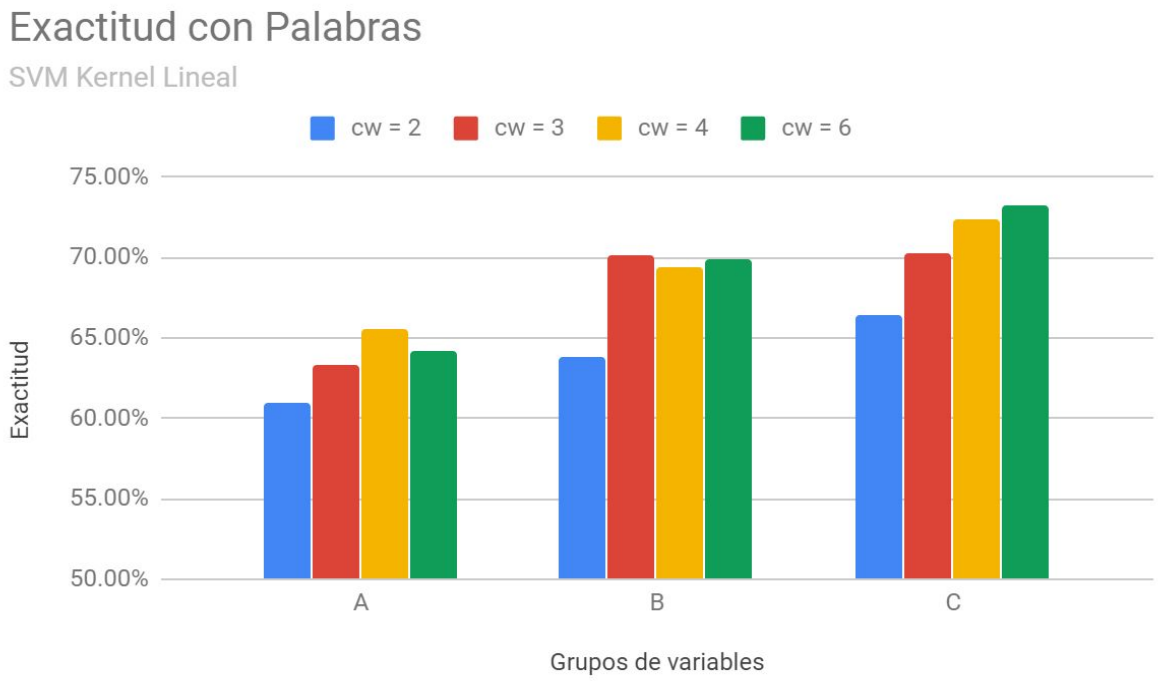


Imagen 34: Exactitud con SVM usando Kernel Lineal, Cost=0.1, tolerance=0.1, usando solo palabras, con distintas cantidades de variables (A,B,C) y valores para cw

Pruebas con **A**,**B**,**C** formados sólo con parejas (**Imagen 35**).

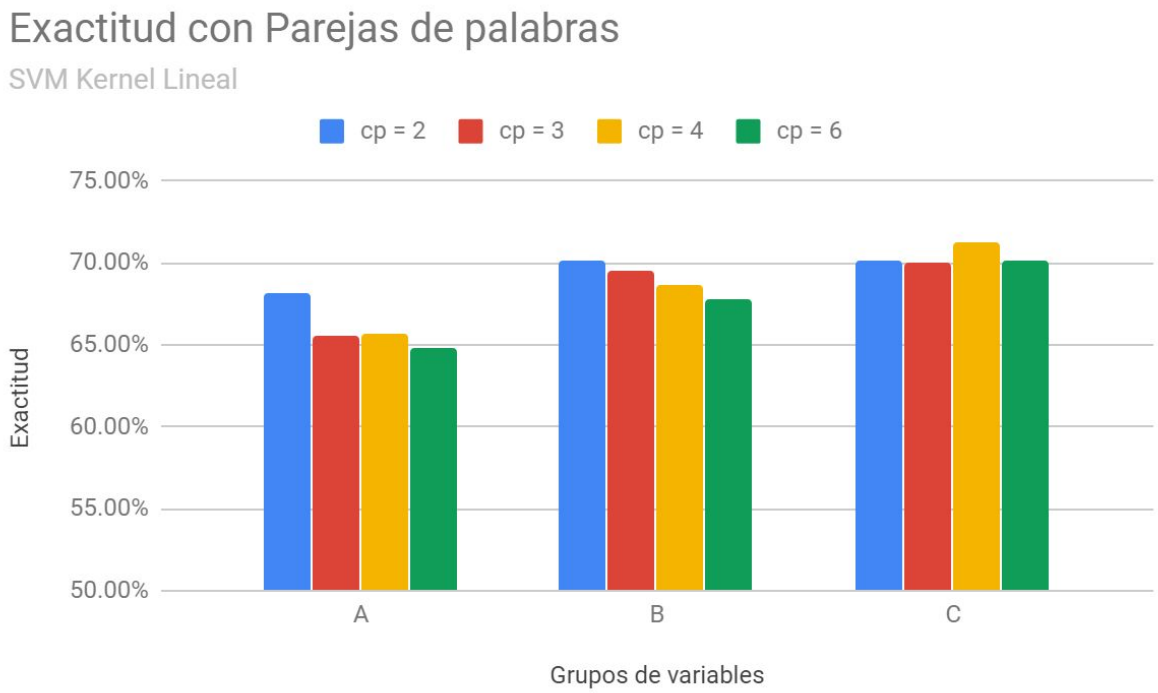


Imagen 35: Exactitud con SVM usando Kernel Lineal, Cost=0.1, tolerance=0.1, usando solo parejas de palabras, con distintas cantidades de variables (A,B,C) y valores para cp

Variables (grupo)	Uso	Exactitud	Ajuste
C	Palabras	73.29%	cw = 6
C	Parejas de palabras	71.20%	cp = 4

Tabla 7: Con SVM Kernel Lineal, mejores resultados de exactitud para Palabras y para Parejas de palabras.

Para comprobar si podemos mejorar los resultados, crearemos un conjunto de datos que contenga los dos mejores conjuntos (**Tabla 7**) y sus respectivos ajustes (*cw,cp)*

El resultado del clasificador y detalles del conjunto de datos (**Imagen 36**)

Variables	6694
Exactitud	74.45%
Uso	Palabras y Parejas
cw - cp	6 - 4
MIN	0.25
MAX	5
MINP	0.3
MAXP	50

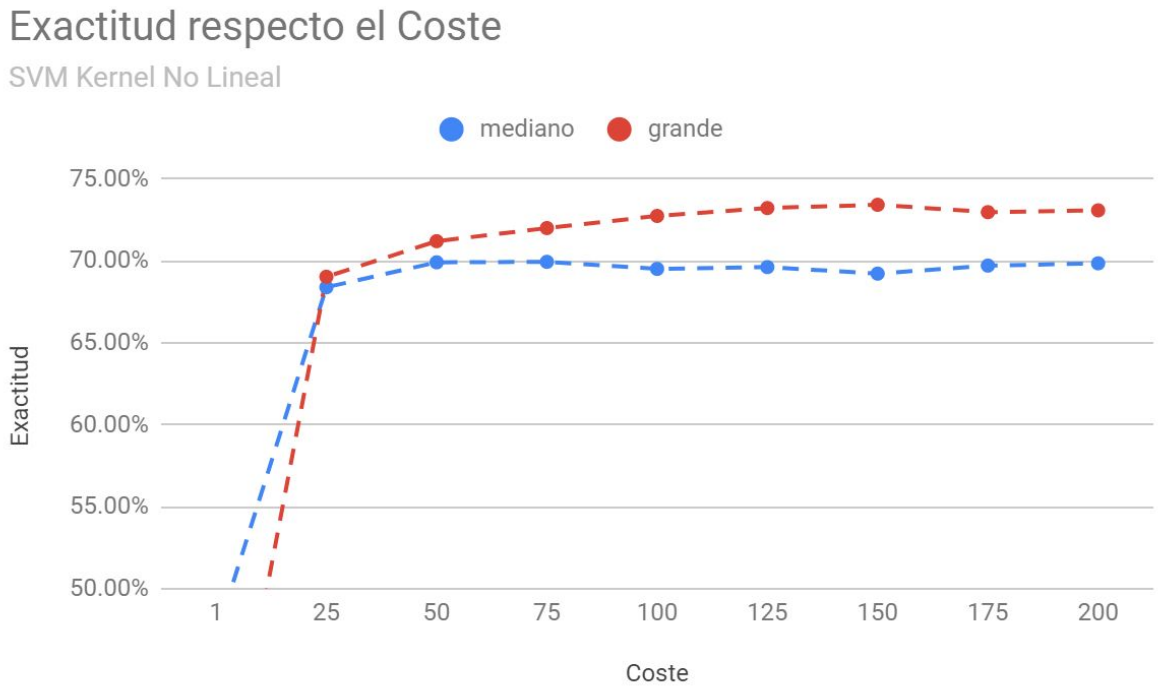
```
       true
pred     PP  Cs PSOE ERC jxcat pod
  PP    555  63   59  13     3  67
  Cs     42 393   21   9     3  27
  PSOE   32  15  280   6     1  23
  ERC    22  16   19 361    74  45
  jxcat   6  24   12  97   504   1
  pod    58  46   39  31     7 474
[1] "Accuracy 0.7445"
```

Imágen 36: Detalles del modelo y exactitud obtenida con SVM Kernel Lineal y la matriz de confusión generada

Afortunadamente, SVM funciona bien con un gran número de variables, a pesar de la enorme cantidad generada (~6700).

Comprobamos que este ajuste de los datos proporciona una **Exactitud del 74.45%**, superior a las obtenidas por los por los conjuntos que lo forman por separado (**Tabla 7**) que consiguen 73.29% y 71.20% respectivamente.

Pruebas con SVM Kernel no Lineal: Base Radial Gaussiana
Con la variable *Cost,* probamos distintos valores para mejorar el modelo.



Imágen 37: Exactitud obtenida con SVM usando Kernel *Base Radial Gaussiana* con los dos conjuntos de variables y distintos valores para *Cost.*

Con las pruebas realizadas (**Imágen 37**) deducimos que el mejor valor, en general, es *Cost = 150*, para conjuntos de datos con suficientes variables(~3000), como el conjunto grande, que ha obtenido una exactitud del 73.42%.
Los resultados del conjunto mediano no varían demasiado al aumentar el coste, por lo que asumimos que el mejor valor de ***Cost* será *150.***

Una vez encontrado el mejor *Cost,* probamos distintos valores para la variable *Tolerance* para mejorar el modelo.
Por defecto *tolerance = 0.001*

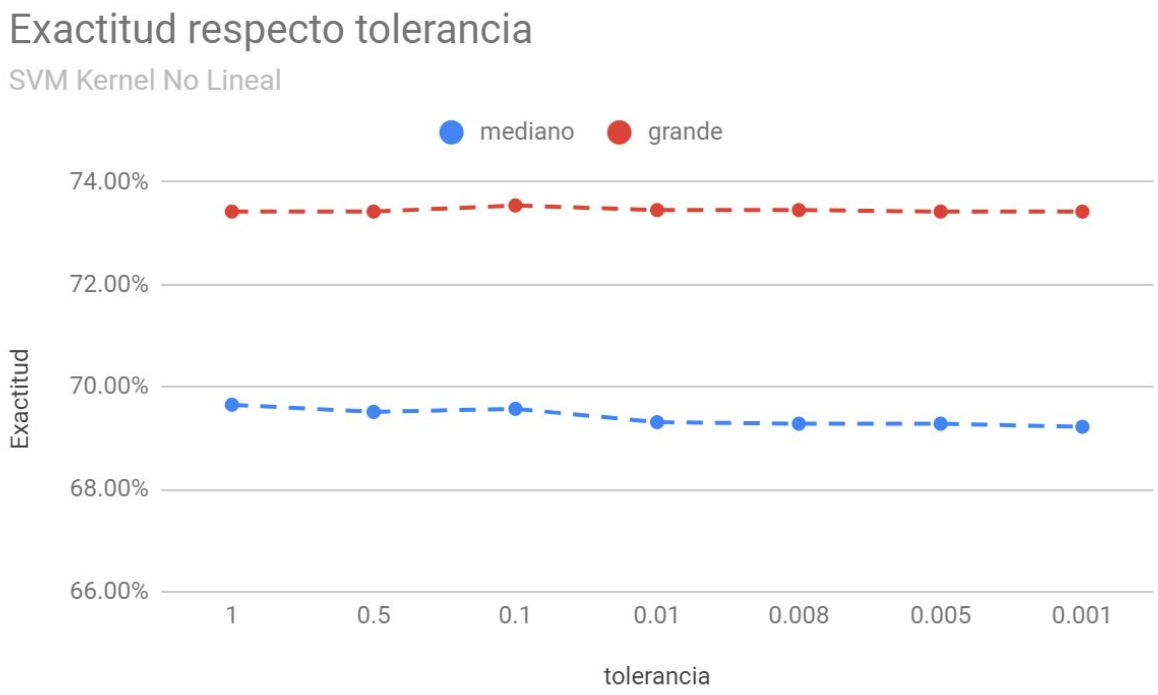


Imagen 38: Exactitud obtenida con SVM usando Kernel *Base Radial Gaussiana, Cost* =150 con los dos conjuntos de variables y distintos valores para *tolerancia.*

Con estas pruebas (**Imágen 38**) se observa que los mejores valores para *tolerance* son *0.1* para conjuntos de datos con suficientes variables, y *0.01* con menos variables.
Sin embargo las diferencias en el resultado al modificar la tolerancia por debajo de 0.1, son mínimas. Asumimos que para, nuestro proyecto, para el *Kernel Base Radial*, el **mejor valor de *tolerance* es *0.1***

También se observa cómo se alcanza el límite de exactitud del sistema, momento en que los resultados ya no varían.

Finalmente probamos ajustando la variable *Gamma.*
Este es el segundo valor necesario (junto con *Cost)* para efectuar la búsqueda en patrón de cuadrícula, necesario en los *Kernels* no lineales

Llevamos a cabo pruebas con diferentes valores, que dependen directamente del número de variables independientes, para ver si con un buen ajuste podemos mejorar los resultados. Por defecto *gamma = 1 / número de Variables.*

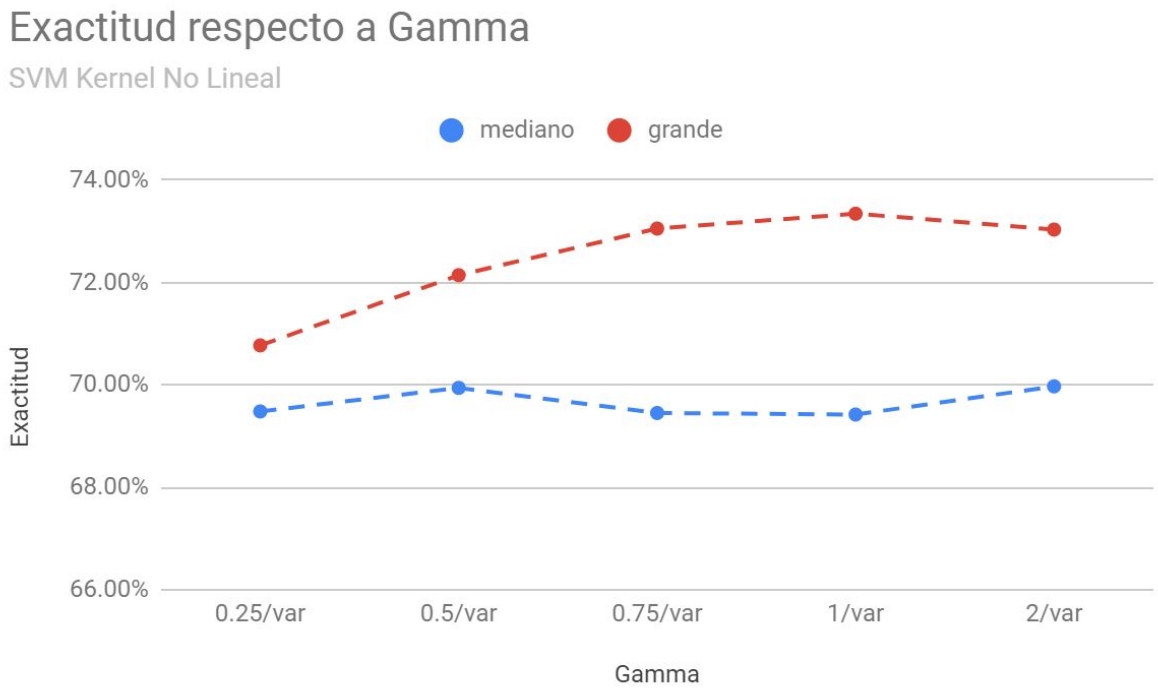


Imagen 39: Exactitud obtenida con SVM usando Kernel *Base Radial Gaussiana, Cost* =150, *tolerance* = *0.1,* con los dos conjuntos de variables y distintos valores para *Gamma*

Estos resultados (**Imágen 39**) nos indican que el valor por defecto (*gamma = 1/num.Var)* ya hace un buen ajuste con grupos de datos grandes, por lo que **dejaremos este valor por defecto**.

Ahora observamos, igual que antes, cómo varía la exactitud con distintas cantidades de variables **A,B,C** (~500, ~1500, ~3500) respectivamente, usando únicamente palabras o parejas. También variamos los valores para *cw, cp*.

Pruebas con **A**,**B**,**C** formados sólo con palabras (**Imagen 40**).

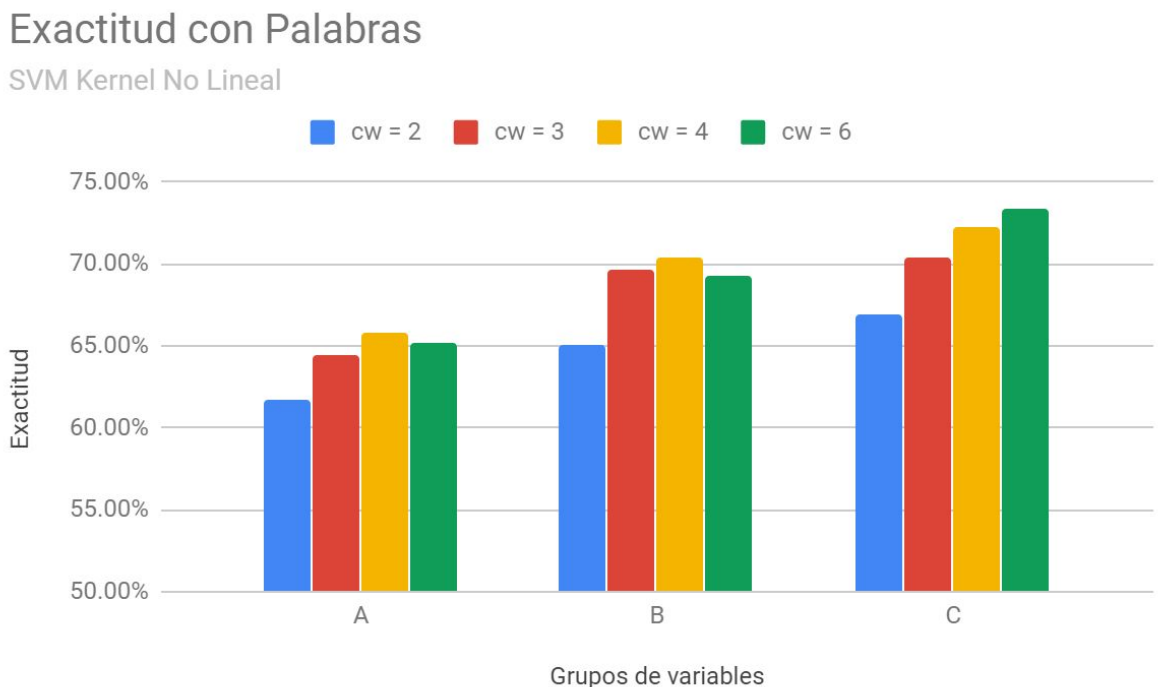


Imagen 40: Exactitud con SVM usando *Kernel Base Radial, Cost=150*, *tolerance=0.1* y usando solo palabras, con distintas cantidades de variables (A,B,C) y valores para *cw.*

Pruebas con **A**,**B**,**C** formados sólo con parejas (**Imagen 41**).

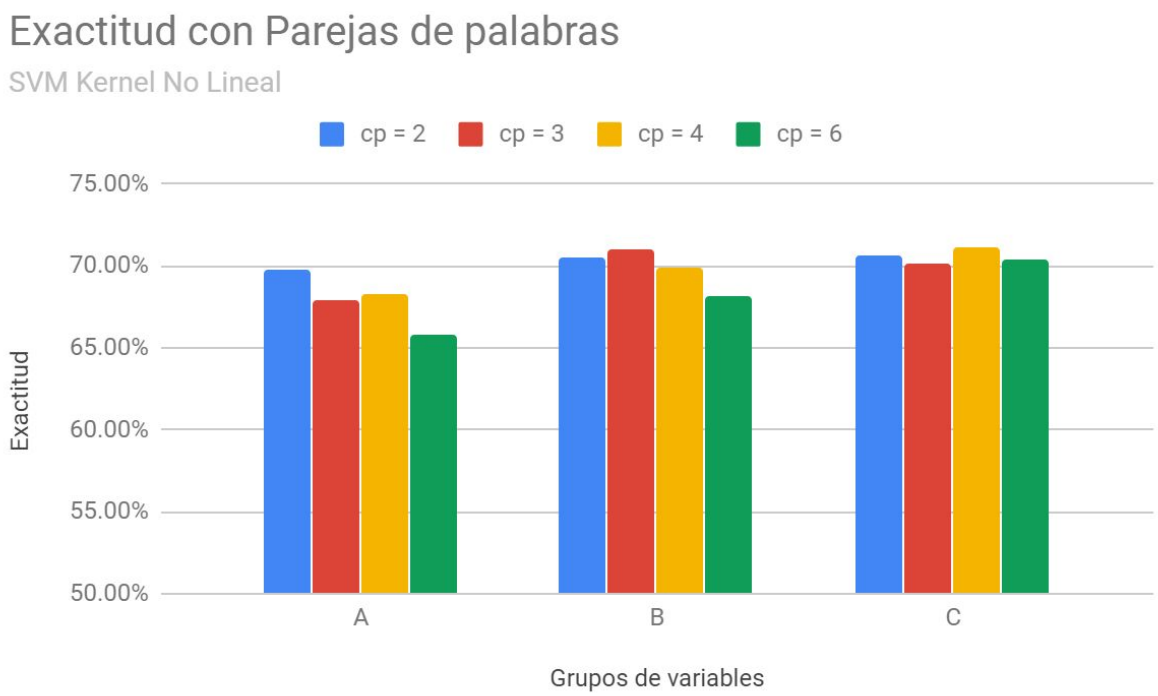


Imagen 41: Exactitud con SVM usando *Kernel Base Radial*, *Cost=150*, *tolerance=0.1* y usando solo parejas de palabras, con distintas cantidades de variables (A,B,C) y valores para *cp*

Los mejores resultados con este *Kernel* (**Tabla 8**) aparecen con los mismos grupos y ajuste de partidos, que en el *Kernel Lineal* (**Tabla 7**) con resultados parecidos.

Variables (grupo)	Uso	Exactitud	Ajuste
C	Palabras	73.34%	cw = 6
C	Parejas de palabras	71.14%	cp = 4

Tabla 8: Con SVM Kernel Base Radial Gaussiana, mejores resultados de exactitud para Palabras y para Parejas de palabras.

Para comprobar si podemos mejorar los resultados, crearemos un conjunto de datos que contenga los dos mejores conjuntos (**Tabla 8**) y sus respectivos ajustes (*cw,cp)*

Resultado del clasificador y detalles del conjunto de datos (**Imagen 39**)

Variables	6694
Exactitud	75.03%
Uso	Palabras y Parejas
cw - cp	6 - 4
MIN	0.25
MAX	5
MINP	0.3
MAXP	50

```
      true
pred    PP  Cs PSOE ERC jxcat pod
  PP   556  60   61  13     3  60
  Cs    42 385   20   5     2  18
  PSOE  24  17  273   3     3  13
  ERC   21  25   24 367    71  46
  jxcat  5  23   12  93   507   1
  pod   67  47   40  36     6 499
[1] "Accuracy 0.7503"
```

Imágen 42: Detalles del modelo y la exactitud obtenida con SVM Kernel Lineal y la matriz de confusión generada

Ahora podemos comparar directamente los resultados con *Kernel Lineal* y *No Lineal*, tratando exactamente los mismos datos.

Comprobamos que *SVM* con *Kernel Base Radial Gaussiana* proporciona una **Exactitud del 75.03%**, superior a la de los conjuntos que la forman, y a la obtenida por el *Kernel Lineal* con los mismos datos.

El mejor modelo en este proyecto es: *Kernel Base Radial Gaussiana*, con **Coste = 150, Gamma = 1/num.var, Tolerance=0.1** y un **conjunto de datos muy grande** (con ~6700 variables independientes).

Conclusiones

En este proyecto hemos desarrollado un sistema para extraer y codificar la información de *tweets,* generar un modelo que aprenda de estos datos y, finalmente, clasificar la ideología política de nuevos *tweets* partiendo de lo que ha aprendido el clasificador con el conjunto de entrenamiento.

Hemos investigado los modelos de *Machine Learning*, y utilizado cuatro de ellos que nos han parecido interesantes para aplicarlos al problema de clasificación.

Estos modelos son ***Linear Discriminant Analysis****,* ***k-Nearest Neighbors****,*
Support Vector Machine Lineal*,* y ***Support Vector Machine Base Radial Gaussiana****.*

Hemos hecho una serie de pruebas con cada clasificador para ajustar los parámetros de este.
Una vez conocidos los mejores ajustes para el modelo, hemos hecho pruebas con distintos conjuntos de datos, para encontrar con que configuración de los datos ofrece mejores resultados la predicción.

El Mejor clasificador:

SVM
Hemos podido comprobar cómo de efectivos son los dos Kernels (lineal/no lineal).
Con ambos se han obtenido buenos resultados, pero con **Kernel *No Lineal Base Radial Gaussiana,*** ajustando bien los parámetros (*Cost, gamma, tolerance)* hemos alcanzando el **75% de exactitud**, lo que sobrepasa muy satisfactoriamente las expectativas iniciales (>51%) y lo convierte en el mejor clasificador en este proyecto.

Referencias

[1] Jason Brownlee (2016). *Understand Machine Learning Algorithms.*

[2] Mitchell, T. (1997). *Machine Learning.*

[3] Michael A. Schuh, Tim Wylie, Rafal A. Angryk. *Mitigating the Curse of Dimensionality for Exact kNN Retrieval.*

[4] Joaquín Amat Rodrigo. *Análisis discriminante lineal (LDA) y Análisis discriminante cuadrático (QDA).*

[5] Bayes, Thomas (1763). *An Essay towards solving a Problem in the Doctrine of Chances.*

[6]Chih-Wei Hsu,Chih-Chung Chang, and Chih-Jen Lin, (2003). *A Practical Guide to Support Vector Classification.*

[7] Vapnik, Chervonekis, (1974). *Structural Risk Minimization principle.*

[8] Dataflair Team (2017). *Kernel Functions-Introduction to SVM Kernel & Examples.*

[9] Thorsten Joachims, (1997). *Text Categorization with Support Vector Machines: Learning with Many Relevant Features.*

[10] Stéphane Canu. *SVM and kernel machines: linear and non-linear classification.*

[11] Yiming Yang, Xin Liu. *A re-examination of text categorization methods.*